

Review Notice

Attached is a copy of the Return-Link Processor (RLP) Peripheral Components Interface (PCI) Card Hardware Definition Document for your review. Please retain this sheet with the document, as it will be used to track subsequent review cycles.

<u>Review Number</u>	<u>Date</u>	<u>To</u>	<u>Date Completed</u>	<u>Comments</u>
--------------------------	-------------	-----------	---------------------------	-----------------

**Return-Link Processor (RLP)
Peripheral Components Interface (PCI) Card
Hardware Definition Document**

August 1997

Responsible Engineer:

Reviewed by:

**K. Winiecki Jr., Lead Engineer
Next Generation Systems Group
Lockheed-Martin Space Mission Systems**

**S. Linehan, Manager
Satellite Ground Systems Group
RMS Technologies, Inc.**

Approved by:

Approved by:

**P. Ghuman, Technical Lead
Next Generation Systems Group
Code 521.0, NASA GSFC**

**N. Speciale, Branch Head
Microelectronic Systems Branch
Code 521, NASA GSFC**

Edited by:

**L. Kane, Sr. Documentation Specialist
NYMA, Inc.**

**Goddard Space Flight Center
Greenbelt, Maryland**

PREFACE

This document is under the configuration management of the Microelectronic Systems Branch Configuration Control Board. Changes to this document shall be made by Documentation Change Notice, reflected in text by change bars, or by complete revision.

Requests for copies of this document, along with questions and proposed changes, should be addressed to:

**Microelectronic Systems Branch, Code 520.9
Goddard Space Flight Center
Greenbelt, Maryland 20771**

DOCUMENT CHANGE INFORMATION

<i>List of Effective Pages</i>		
Page Number		Issue
i		Original
1-1 thru		Original
2-1 thru		Original
3-1 thru		Original
4-1 thru		Original
5-1 thru		Original
6-1 thru		Original
AB-1 thru		Original
A-1 thru		Original
B-1 thru		Original
C-1 thru		Original
D-1 thru		Original
E-1 thru		Original
<i>Document History</i>		
Issue	Date	Document Control Number
Original	March 1997	N/A

TABLE OF CONTENTS

SECTION 1 - GENERAL INFORMATION	1-1
1.1 Purpose.....	1-1
1.2 Scope.....	1-1
1.3 General Description.....	1-1
1.4 Reference Documents.....	1-3
SECTION 2 - FUNCTIONAL DESCRIPTION.....	2-1
2.1 Introduction.....	2-1
2.2 Theory of Operation.....	2-2
2.3 Modes of Operation.....	2-3
2.4 Functional Elements.....	2-3
2.4.1 PCI Interface.....	2-3
2.4.2 Input Interface.....	2-4
2.4.3 Parallel Integrated Frame Synchronizer.....	2-5
2.4.4 Reed-Solomon Error Detector/Corrector.....	2-5
2.4.5 CCSDS Service Processor.....	2-5
2.4.6 Status Collector	2-7
2.4.7 DMA Interface.....	2-8
2.4.8 Baseboard Interface.....	2-9
2.5 Interrupts.....	2-9
SECTION 3 - MEMORY MAP AND REGISTER DEFINITIONS	3-1
3.1 Introduction.....	3-1
3.2 Memory Map.....	3-1
3.3 Memory Map Description and Register Definitions.....	3-2
3.3.1 DMA-Chaining/User Memory \$00 0000	3-2
3.3.2 Baseboard Interface Registers \$04 0000.....	3-2
3.3.2.1 Control Register \$04 0000.....	3-2
3.3.2.2 Interrupt Source Mask Register A \$04 0004.....	3-3
3.3.2.3 Interrupt Source Mask Register B \$04 0008.....	3-3
3.3.2.4 Interrupt Source Register A \$04 000C.....	3-4
3.3.2.5 Interrupt Source Register B \$04 0010	3-5
3.3.3 DMA Interface Transfer FIFO Output Space \$08 0000.....	3-6
3.3.4 DMA Interface Controller \$0A 0000	3-6
3.3.4.1 Channel A Control Register \$0A 0000.....	3-6
3.3.4.2 Channel B Control Register \$0A 0004.....	3-6
3.3.4.3 Channel A Fill Count Register \$0A 0008	3-7
3.3.4.4 Channel B Fill Count Register \$0A 000C.....	3-7
3.3.4.5 General Control Register \$0A 0010.....	3-8
3.3.5 DMA Interface Data FIFO Output Space \$0C 0000.....	3-8
3.3.6 DMA Interface Transfer FIFO Input Space \$0E 0000.....	3-8
3.3.7 Status Collector \$10 0000.....	3-9
3.3.7.1 Status Request Register \$10 0000.....	3-9
3.3.7.2 Analog Values Register \$10 0004.....	3-9
3.3.8 Status Data FIFO Input Space \$10 8000.....	3-9
3.3.9 PIFS Register Space \$11 0000.....	3-10
3.3.10 RS Register Space \$12 0000.....	3-10
3.3.11 RS Memory Space \$13 0000.....	3-10
3.3.12 SP Register Space \$14 0000.....	3-10
3.3.13 SP Frame Status Memory Space \$15 0000	3-10
3.3.14 SP Packet Status Memory Space \$16 0000	3-11

3.3.15 SP Internally-Accessible Memory Space \$18 0000..... 3-11

TABLE OF CONTENTS (CONTD)

SECTION 4 - HARDWARE INSTALLATION.....	4-1
4.1 Introduction.....	4-1
4.2 Hardware Elements.....	4-1
4.2.1 I/O Panel.....	4-1
4.2.2 Internal Connectors.....	4-1
4.2.2.1 MACHPRO/JTAG Connector J6.....	4-2
4.2.2.2 Optional Sorting Module Connector J7.....	4-2
4.2.2.3 PCI Connector P1.....	4-2
4.2.3 Jumpers and Switches	4-3
4.3 Interfaces.....	4-3
4.3.1 LED	4-3
4.3.2 ECL Data Input Interface.....	4-4
4.3.3 RS-422/485 Data Input Interface	4-4
4.3.4 External Timecode Reference Clock Input Interface.....	4-4
4.3.5 MACHPRO/JTAG Interface.....	4-4
4.3.6 Optional Sorting Module Interface.....	4-5
4.3.7 PCI Bus Interface.....	4-6
4.4 Installation Guidelines.....	4-7
4.5 Environmental Requirements	4-7
SECTION 5 - OPERATING PRINCIPLES	5-1
5.1 Introduction.....	5-1
5.2 Preliminaries	5-1
5.3 Setup.....	5-1
5.3.1 Resets and Inputs.....	5-1
5.3.2 PCI Interface.....	5-1
5.3.3 Input Interface.....	5-2
5.3.4 PIFS.....	5-2
5.3.5 Data Routing Mode.....	5-2
5.3.6 RS	5-2
5.3.7 SP.....	5-2
5.3.8 DMA Interface.....	5-3
5.3.9 Status Collector	5-3
5.3.10 Interrupts.....	5-3
5.3.11 Programmable FIFO Flags.....	5-3
5.3.12 Baseboard Interface.....	5-4
5.4 Operation.....	5-5
5.4.1 Wait for Data	5-5
5.4.2 Initiate a Transfer.....	5-5
5.4.3 Wait for DMA Completion	5-6
5.4.4 Recheck for Data	5-6
5.4.5 Flush Remaining Data.....	5-6
SECTION 6 - SCHEMATIC DESCRIPTION.....	6-1
6.1 Introduction.....	6-1
6.2 Drawing Directory.....	6-1
6.3 Drawing Description.....	6-1
6.3.1 Top Assembly.....	6-1
6.3.2 Bottom Assembly.....	6-1
6.3.3 Drill Master.....	6-2
6.3.4 Sheet 1.....	6-2

6.3.5 Sheet 2.....6-2
6.3.6 Sheet 3.....6-2
6.3.7 Sheet 4.....6-2
6.3.8 Sheet 5.....6-2
6.3.9 Sheet 6.....6-2

TABLE OF CONTENTS (CONTD)

6.3.10	Sheet 7.....	6-2
6.3.11	Sheet 8.....	6-2
6.3.12	Sheet 9.....	6-2
6.3.13	Sheet 10.....	6-2
6.3.14	Sheet 11.....	6-3
6.3.15	Sheet 12.....	6-3
6.3.16	Sheet 13.....	6-3
6.4	Program Listing Directory.....	6-3
6.5	Program Listing Description.....	6-3
6.4.1	Baseboard Interface.....	6-3
6.4.2	DMA Interface.....	6-3
6.4.3	Status Collector	6-4
6.4.4	PCI Configuration ROM.....	6-4
ABBREVIATIONS AND ACRONYMS.....		AB-1
APPENDIX A - BOARD LAYOUT AND SCHEMATICS.....		A-1
APPENDIX B - PROGRAMMABLE DEVICE LISTINGS		B-1
APPENDIX C - PARTS LIST.....		C-1
APPENDIX D - ENGINEERING CHANGE ORDERS.....		D-1
APPENDIX E - TEST PROCEDURES		E-1

LIST OF TABLES.C.LIST OF TABLES;

Table 3-1	RLP Memory Map.....	3-1
Table 3-2	Baseboard Interface Control Register \$04 0000.....	3-2
Table 3-3	Baseboard Interface Interrupt Source Mask Register A \$04 0004.....	3-3
Table 3-4	Baseboard Interface Interrupt Source Mask Register B \$04 0008.....	3-3
Table 3-5	Baseboard Interface Interrupt Source Register A \$04 000C.....	3-4
Table 3-6	Baseboard Interface Interrupt Source Register B \$04 0010.....	3-5
Table 3-7	DMA Interface Channel A Control Register \$0A 0000.....	3-6
Table 3-8	DMA Interface Channel B Control Register \$0A 0004.....	3-7
Table 3-9	DMA Interface Channel A Fill Count Register \$0A 0008.....	3-7
Table 3-10	DMA Interface Channel B Fill Count Register \$0A 000C	3-7
Table 3-11	DMA Interface General Control Register \$0A 0010.....	3-8
Table 3-12	Status Collector Status Request Register \$10 0000	3-9
Table 3-13	Status Collector Analog Values Register \$10 0004.....	3-9
Table 4-1	RLP I/O Panel Interfaces.....	4-3
Table 4-2	RLP MACHPRO/JTAG Connector J6 Signals.....	4-4
Table 4-3	RLP Optional Sorting Module Connector J7 Signals.....	4-5
Table 4-4	RLP PCI Bus Connector P1 Signals	4-6
Table 6-1	RLP Drawing Directory.....	6-1
Table 6-2	RLP Program Listing Directory.....	6-3

LIST OF FIGURES.C.LIST OF FIGURES;

Figure 1-1	Code 521 Current Generation Return-Link Data Processing System.....	1-1
Figure 1-2	Code 521 Next Generation Return-Link Processor PCI Card.....	1-2
Figure 1-3	Code 521 Next Generation Desktop Satellite Data Processor.....	1-2
Figure 2-1	Return-Link Processor PCI Card Functional Block Diagram	2-1
Figure 2-2	RLP PCI Card & Optional Sorting Module Detail.....	2-4
Figure 2-3	RLP Service Processor Detail.....	2-6
Figure 2-4	RLP Status Collector & Baseboard Interface Detail.....	2-7
Figure 4-1	RLP I/O Panel	4-1
Figure 4-2	RLP Internal Connector Placement.....	4-2
Figure 4-3	RLP MACHPRO/JTAG Connector J6.....	4-2
Figure 4-4	RLP Optional Sorting Module Connector J7.....	4-2
Figure 4-5	RLP PCI Connector P1	4-3

FEATURES

- Performs all telemetry return-link data processing functions for a ground station.
- Works in common Peripheral Components Interface (PCI)-compliant host computers.
- 0 to 300 Mbps nominal telemetry data input rate, 400 Mbps theoretical maximum.
- Emitter-coupled logic (ECL) differential, ECL single-ended, and RS-422/RS-485 bit-serial telemetry data input.
- Byte-serial, bit-parallel telemetry data input direct from host computer.
- Optional timecode reference clock signal input from external source.
- Performs all functions of a telemetry data frame synchronizer (FS) on both Consultative Committee for Space Data Systems (CCSDS) and non-CCSDS (Weather, etc.) data
- Performs all functions of a Reed-Solomon Error Detector/Corrector (RS) on frame-synchronized data
- Performs all functions of a CCSDS Packet Telemetry and Advanced Orbiting Systems (AOS), Networks, and Data Links Service Processor (SP) on frame-synchronized data.
- Sorts user-definable packet-level data into up to four packet service buffers, and sorts user-definable frame-level data into up to eight frame service buffers.
- Data queued up in 32-bitwide transfer buffer for maximum PCI burst rate of 1 Gbps.
- Interrupt-driven or polled service methods.
- Request-driven automatic status collection and buffering.
- Separate, isolated status and telemetry data busses.
- Digital power consumption (current draw) and ambient temperature monitoring.
- All board logic reprogrammable in-circuit.
- No onboard microprocessing unit (MPU), heat sinks, jumpers, or switches.
- PCI Joint Test Action Group (JTAG) boundary scan for in-system testing.
- Expandable via optional mezzanine board for additional/custom functionality.

PHYSICAL DESCRIPTION

- Universal voltage (5 or 3.3 V), 32-bit, 33 MHz, full-height, full length PCI Expansion Card.
- One input/output (I/O) panel light-emitting diode (LED.)
- Four I/O panel sub-miniature "B" (SMB) jacks for differential or single-ended ECL serial data and clock signal input.
- One I/O panel DB-9 jack for RS-422/RS-485 data and clock signal input and optional 10 MHz transistor-transistor logic (TTL) timecode reference clock signal input.
- On-board timecode reference clock oscillator.
- Three National Aeronautics and Space Administration (NASA) Goddard Space Flight Center (GSFC) Mission Operations and Data Systems Directorate (MO&DSD) Data Systems Technology Division (DSTD) Microelectronic Systems Branch (MSB) (Code 521) custom application-specific integrated circuits (ASICs):
 - Parallel, Integrated Frame Synchronizer (PIFS).
 - Reed-Solomon Error Detector/Corrector (RS).
 - CCSDS Packet Telemetry/AOS Service Processor (SP).
- Commercial off-the-shelf (COTS) PCI Local Bus interface ASIC.
- Three large "complex" programmable logic devices (CPLD) in-circuit-reprogrammable through onboard connector.
- One 200-pin connector for optional, variable-sized, custom-function mezzanine board and test signal breakout.

SECTION 1 GENERAL INFORMATION

1.1 PURPOSE

This document describes the Return-Link Processor (RLP) Peripheral Components Interface (PCI) Card part number G1527437, developed by the NASA GSFC Code 521 Next Generation Systems (NGS) group.

1.2 SCOPE

This document provides a functional description, memory map, register models, processing scenario, and schematic description for the RLP. It is assumed that the reader is familiar with the specification and operation of: PCI Local Bus, Systems and Expansion Cards; V3 Corporation V962 PCI Bridge Chip (PBC) PCI Local Bus Bridge application specific integrated circuit (ASIC); Consultative Committee for Space Data Systems (CCSDS) Packet Telemetry and Advanced Orbiting System (AOS) Services; NASA Communications (Nascom) data signals; telemetry return-link data processing functions including frame synchronization, Reed-Solomon error detection/correction, and CCSDS Packet Telemetry and AOS Service Processing; and NASA GSFC Code 521 ASICs including the Parallel, Integrated FS (PIFS), Reed-Solomon Error Detector/Corrector (RS), and CCSDS Packet Telemetry and AOS Service Processor (SP).

1.3 GENERAL DESCRIPTION

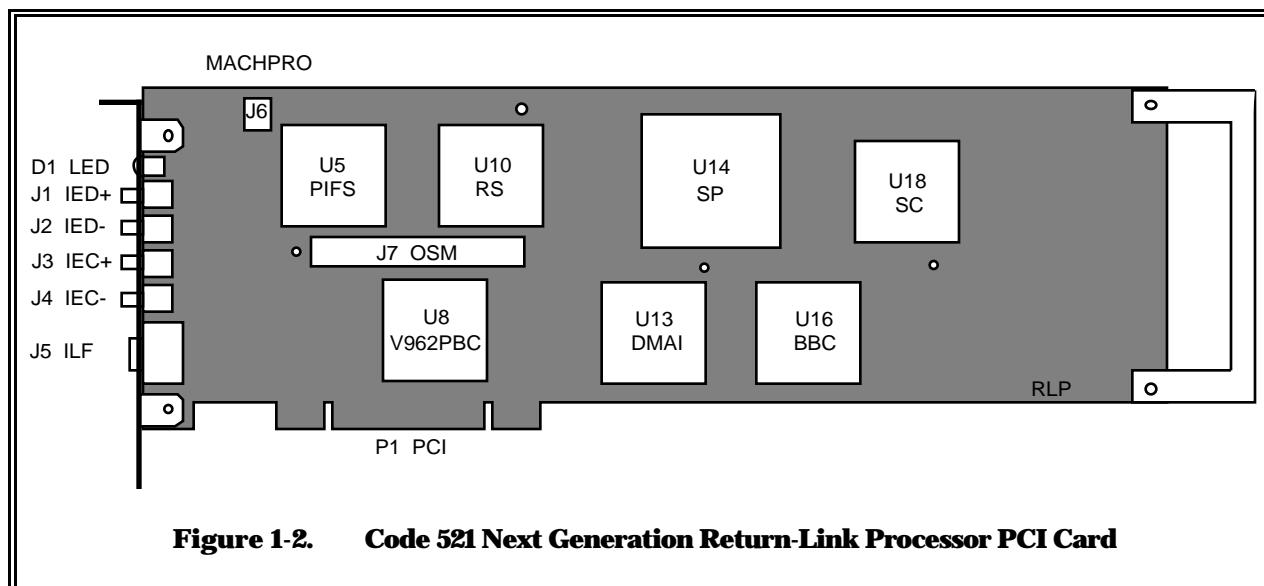
The current generation of Code 521-developed telemetry return-link data processing systems requires a number of Versa-Module Euroboard (VME) cards, custom ASICs, and embedded microprocessing units (MPUs) to perform high-speed (>2 Mbps) real-time return-link data processing. In particular, frame synchronization at 50 Mbps and below is performed by a VME card containing a Code 521 complimentary metal-oxide-semiconductor (CMOS) Frame Synchronizer (FS) ASIC and support circuitry; frame synchronization above 50 Mbps is performed by a VME card containing a Code 521 Gallium-Arsenide (GaAs) FS ASIC and support circuitry; Reed-Solomon error detection and correction at up to 528 Mbps are performed by a VME card containing a Code 521 CMOS RS ASIC and support circuitry; and CCSDS Packet Telemetry and AOS Service processing at up to 50 Mbps is performed by a VME card containing two Code 521 CMOS telemetry data pipeline ASICs, three Motorola 68040 MPUs, and support circuitry (see Figure 1-1).

[to be added by Technical Publications]

Figure 1-1. Code 521 Current Generation Return-Link Data Processing System

Continuing the evolution of telemetry processing components towards smaller/cheaper/faster by applying the current state-of-the-art in microelectronic technology, the Code 521 Next Generation Systems (NGS) group recently parallelized and integrated the FS and SP functions into two CMOS ASICs capable of up to 528 Mbps (matching the RS). The RLP is the next generation return-link data processing solution, the combination of the three RLP functions into a single board-level product. It is designed to operate from 0 to 300 Mbps and can theoretically be operated up to 400 Mbps. It is also the first application of the Code 521 SP ASIC, as well as the first application of the Code 521 PIFS ASIC above 10 Mbps.

The RLP is a fully Plug and Play (P&P)-compliant PCI Expansion Card capable of operating in all PCI host computers: International Business Machines (IBM)-compatibles, Macintosh-compatibles, Digital Equipment Corporation (DEC) stations, Sun SPARCstations, etc. See Figure 1-2. Thus, the expense, complexity, and possible concerns about the aging VME platform are eliminated.



Serial telemetry return-link data is most commonly the output product of a bit-synchronizer, either in local receiver hardware or at Nascom; it could also be generated by test hardware or played back from storage. This enters the PIFS directly through the connectors on the I/O panel. The host computer can inject bit-parallel, byte-serial data at relatively low rates directly into the PIFS via the PCI bus, so telemetry data can also be generated by the host or obtained over a network.

The processed telemetry data accumulates in first-in, first-out (FIFO) memory buffers for driver software running on the host computer to transfer out over the PCI bus, generally into main memory, where it is further processed or transferred to another PCI card, into storage, or out across a network. The RLP can also accept an optional mezzanine board which can conceivably be developed to output data directly to a storage or network device.

Figure 1-3 shows how the RLP fits into the Next Generation Desktop Satellite Data Processor (DSDP), the logical future of telemetry processing.

[to be added by Technical Publications]

Figure 1-3. Code 521 Next Generation Desktop Satellite Data Processor

The PIFS performs CCSDS as well as custom frame synchronization, cyclic redundancy code (CRC)/bit transition density (BTD) decoding, CCSDS day-segmented timecode generation, and frame annotation. Data from the PIFS can be output via the PCI bus, as would be done with Weather data, or input to the RS, which would be the case for most CCSDS data. The RS performs Reed-Solomon error detection and correction and frame quality annotation. Data from the RS can also be output via the PCI bus or sent to the SP. The SP performs CCSDS Packet Telemetry and AOS Service Processing, including frame service and packet quality annotation. It sorts frame

components and annotation into up to eight frame service buffers, and packet components and annotation into up to four packet service buffers, for output via the PCI bus. The host also has access to a status data buffer which can contain all of the processing status and data quality information needed for real-time monitoring.

The RLP has a connector for an optional mezzanine board. The board can be designed to perform additional manipulation of the output data such as buffering, sorting, processing, annotating, testing, and output.

Support for the RLP on particular PCI platforms depends not on the hardware but driver software development. Driver software is currently being developed for the Intel Pentium/Microsoft Windows-NT and DEC Alpha/DEC Ultrix platforms.

Physically, the RLP is a universal voltage (5 or 3.3 V), 32-bit, 33 MHz, full-height, full length PCI Expansion Card. Its I/O panel contains one host-operated LED, four sub-miniature "B" (SMB) jacks for differential or single-ended emitter-coupled logic (ECL) serial data and clock signal input, and one DB-9 jack for RS-422/RS-485 data and clock signal input and optional 10 MHz TTL timecode reference clock signal input. The card contains an onboard timecode reference clock signal oscillator, three Code 521 custom ASICs (PIFS, RS, SP), a COTS PCI/local bus bridge ASIC, three large in-circuit-reprogrammable "complex" programmable logic devices (CPLD) and a programming connector, and one 200-pin connector for an optional mezzanine board or test signal breakout. See Figure 1-2.

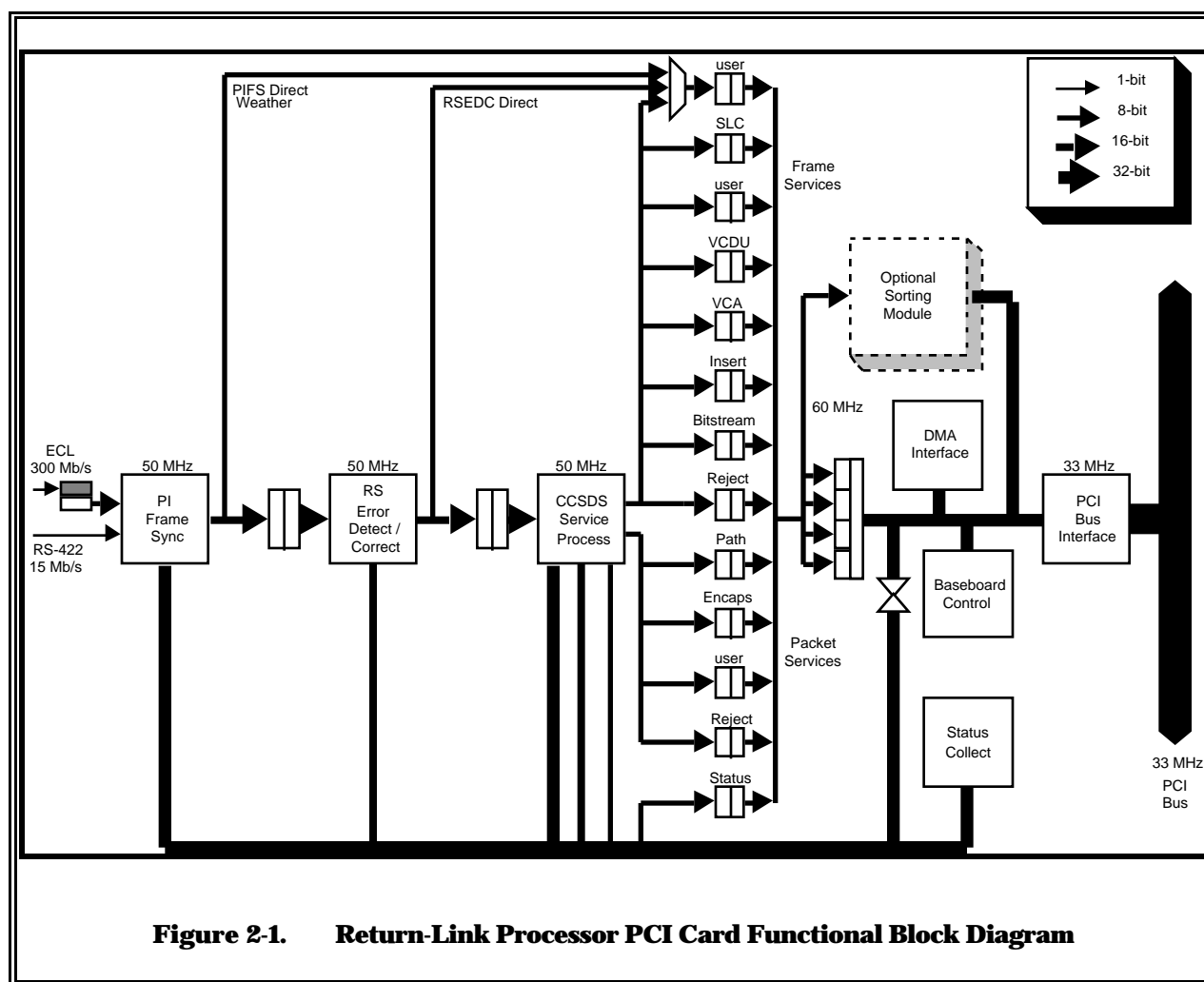
1.4 REFERENCE DOCUMENTS

- Packet Telemetry. CCSDS 102.0-B-3, CCSDS, Annapolis MD.
- Advanced Orbiting Systems, Networks, and Data Links: Architectural Specification. CCSDS 701.0-B-2, CCSDS, Annapolis MD.
- Parallel Integrated Frame Synchronizer Chip. 521-ASIC-023, NASA GSFC, Greenbelt MD.
- Microelectronic Systems Branch Application-Specific Integrated Circuits (ASIC) Components Document. 521-SPEC-002, NASA GSFC, Greenbelt MD.
- Service Processor Chip. 521-ASIC-???, NASA GSFC, Greenbelt MD.
- PCI Local Bus Specification Revision 2.1. PCI Special Interest Group, Portland OR.
- VxxxPBC User's Manual Revision 2.0. V3 Semiconductor Corp., Santa Clara CA.
- AT24C01A/2/4/8/16 2-Wire Serial CMOS E²PROM Data Sheet. ATMEL, San Jose CA.
- MACH466/MACH466LV466-10/12/15 High-Density EE CMOS Programmable Logic Data Sheet. Advanced Micro Devices (AMD) Inc., Sunnyvale CA.
- MACHPRO(TM) Programming Software Manual for AMD MACH JTAG Devices Version 1.3x. Advanced Micro Devices Inc., Sunnyvale CA. (MACHPRO is a registered trademark of Advanced Micro Devices, Inc.)
- IDT72261/72271 CMOS SuperSync FIFO Data Sheet. Integrated Device Technology, Santa Clara CA.

SECTION 2 FUNCTIONAL DESCRIPTION

2.1 INTRODUCTION

The RLP performs the satellite telemetry return-link data processing functions in real time up to a nominal 300 Mbps with a theoretical maximum of 400 Mbps. Functions include frame synchronization, Reed-Solomon error detection and correction, and CCSDS Packet Telemetry and AOS Service Processing, all on a single industry-standard PCI Expansion Card. In a typical scenario, a standard serial data stream is connected through the I/O panel, processed in dataflow-fashion through the Code 521 custom ASICs, and deposited in a number of FIFO memory buffers for software running on the PCI host computer to transfer elsewhere; see Figure 2-1. Weather Satellite Data and other formats can be frame synchronized and routed around the SP and/or RS functions. Internet and other low-rate computer-accessible data sources can be injected directly by the host via the PCI bus. The RLP also contains a connector for an Optional Sorting Module, a variable-sized mezzanine board on which additional processing, buffering, and output functions can be implemented.



This section provides a functional description of the RLP.

2.2 THEORY OF OPERATION

The RLP functions as a PCI slave device to the host computer (master). It is seen by application software as an area of memory that can be written (setup, control, data input) and read (status, data output); to see the memory map, refer to Table 3-1. The RLP has no onboard MPU, so software running on the host is responsible for setup, control, monitoring, and transferring out the processed data. The card can generate interrupts to the host to request service, or its status can be polled; overall function is the same in either case.

Upon host initialization, the host operating system and driver software interact with the RLP as specified by the PCI Specification to configure the device and the low-level host interface. Eventually, application software sets up the card for a "data session". Setup information ultimately comes from the user; the host sets up each of the RLP components (described in Section 5.3) by writing to the appropriate areas of memory. The system is ready to accept telemetry data as soon as setup is complete.

Telemetry data commonly enters the card in the form of a serial bitstream through connectors on the I/O panel. It can also be injected by the host via the PCI bus in bit-parallel, byte-serial form (one byte per 32-bit word write); this could be used for Internet, test, or other very low rate data. The data goes directly into the PIFS. Output from the PIFS goes either to the RS or to a data FIFO per host setup. If set up to use the RS, its output goes either to the SP or to a data FIFO, per host setup. If set up to use the SP, its output goes in up to 12 data FIFOs, per host setup. These three data flow options are mutually exclusive and remain fixed within a data session.

During a data session, the host may want to obtain the status of not just the immediate data flow operation, but of each processing element on the RLP, the quality of the data itself, or other hardware conditions such as ambient temperature or total current draw (power consumption). This is accomplished by writing a status request code into a status request register; the RLP then collects all the various status information requested and places it into a dedicated data FIFO without further host interaction and without interfering with the flow of telemetry data.

As data enters the thirteen data FIFOs (twelve SP and one Status), the fullness of each FIFOs is recorded in a pair of baseboard condition registers, along with whether a Status Collection Operation has completed and whether the FIFO receiving data from the PIFS has overflowed. These registers can be polled by the host or they can be set up to cause an interrupt, at which point the host reads them to determine the interrupt source.

Most of the conditions tell the host that at least some known quantity of data is ready to be transferred out. The host typically transfers the data to host memory. To do this, the host first sets up the byte transfer with the Direct Memory Access (DMA) Interface (DMAI), which is responsible for moving data out of the 8-bit-wide data FIFOs and packing it into a 32-bit-wide transfer FIFO, and then sets up the DMA in the PCI Bridge Chip (PBC), which is responsible for reading the data from the transfer FIFO across the PCI bus to the specified destination. At this point any previously-requested status data is handled in the same manner as the telemetry data, providing a uniform data transfer interface throughout the data session. The host is responsible for prioritizing the order in which the data FIFOs are serviced and the amount of data buffered before the need for service is indicated. Other processes running on the host typically transfer the data from host memory to storage or to another host across a network and might also perform additional data processing.

At the end of a data session, some data will likely remain in the data FIFOs; flushing these is as straightforward as a normal data transfer. If the SP was set up to process split packets, there may

also be unused pieces that must be manually extracted as described in the SP document listed in Section 1.4.

After flushing, the RLP may be set up for a new session, or depending on how it was previously set up, may be ready to begin another session immediately.

2.3 MODES OF OPERATION

The RLP has only one mode of operation; this is what was described in Section 2.2. Other modes may be added by the Optional Sorting Module such as the ability to act as the PCI bus master, perform automated self-testing or control, or output telemetry data directly to an I/O device; refer to the hardware definition document for the specific module of interest for more information.

2.4 FUNCTIONAL ELEMENTS

The functional elements of the RLP consist of: the PCI Interface, the Input Interface, the Parallel Integrated Frame Synchronizer, the Reed-Solomon Error Detector/Corrector, the CCSDS Service Processor, the Status Collector, the Baseboard Interface, and the DMA Interface.

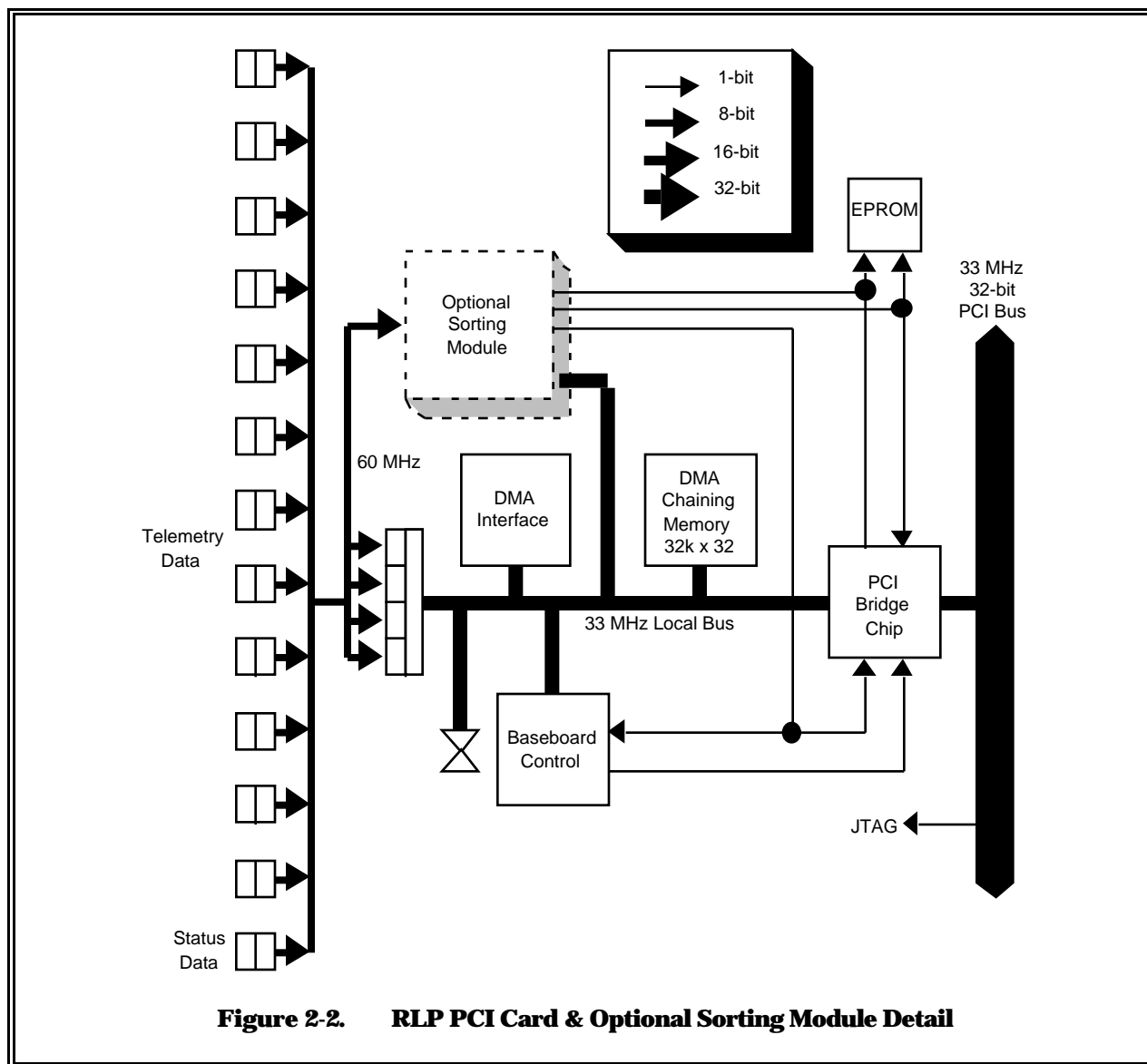
2.4.1 PCI INTERFACE

The RLP conforms to the industry standard PCI Local Bus Specification Revision 2.1; see Section 1.4 for the complete reference. It is a full-height, full-length, 33 MHz, 32-bit data, universal-voltage PCI expansion device. PCI is "Plug & Play" (P&P), so the RLP should function in any PCI-compliant system for which driver software has been provided. During a data burst, the PCI bus is capable of transferring one 32-bit word every 33 MHz clock cycle for a throughput of over 1 Gbps. It receives the 33 MHz PCI bus clock from the host computer.

The local bus is a 33 MHz, fully-synchronous, 32-bit demultiplexed address and data bus. It has the signalling and timing characteristics necessary to interface directly with the Intel i960Cx/Hx series of microprocessors, although the RLP itself uses no MPU. It requires the local bus clock to be provided.

PCI provides JTAG Boundary Scan signals that are connected to the SP and PIFS ASICs to facilitate automated testing.

Connectivity is provided by the V962PBC PCI Bus Bridge Chip from V3 Corporation; see Figure 2-2. It provides dual, chainable DMA engines, and the RLP contains 128 Kbytes of 32-bit-wide local memory to hold the chaining list. This is more than the chaining list requires, so the additional memory may be used by the host as desired. The PBC has been wired to generate a PCI interrupt from two sources, the Baseboard Interface and the Optional Sorting Module. However, the Baseboard Interface can also generate the interrupt for the Optional Sorting Module, so the user can decide which mode is preferable. The PBC has been wired to automatically load its configuration-space from an onboard ROM. The ROM is in-circuit reprogrammable through the PBC; more information is available from the reference documents listed in Section 1.4. If an Optional Sorting Module is present, the onboard ROM is automatically replaced by one on the Module. In this way, all Modules can use the same version of the RLP.



2.4.2 INPUT INTERFACE

The I/O panel contains four standard SMB jacks for ECL differential or single ended data and clock signals up to 300 Mbps nominal, 400 Mbps theoretical maximum with the 50 MHz processing clock (the maximum allowed by the ECL circuit). It also contains one DB-9 jack for low-rate (10 Mbps) differential RS-422 or RS-485 data and clock signals, although past experience with the circuits indicates that up to 25 Mbps is often possible. The ECL inputs are terminated to 50 Ω into $-2V$ and the low-rate telemetry inputs are terminated to 120 Ω . Signals may be connected to both ECL and low-rate inputs simultaneously, but only one will be used for the data source as determined by setup from the host; refer to Section 4 for details.

ECL data is parallelized into bytes by onboard circuitry for input to the PIFS. Low-rate data is slow enough to enter the PIFS directly in serial form. A single-ended ECL clock signal requires special accommodation; see Section 4.3.2 for details.

There is also an external input for a high-accuracy TTL-level 50 ± 10 MHz reference clock that the PIFS can use for timecode generation in lieu of the onboard oscillator. Use of this input is selected by setup from the host and if left connected while unused will have no adverse effect.

2.4.3 PARALLEL INTEGRATED FRAME SYNCHRONIZER

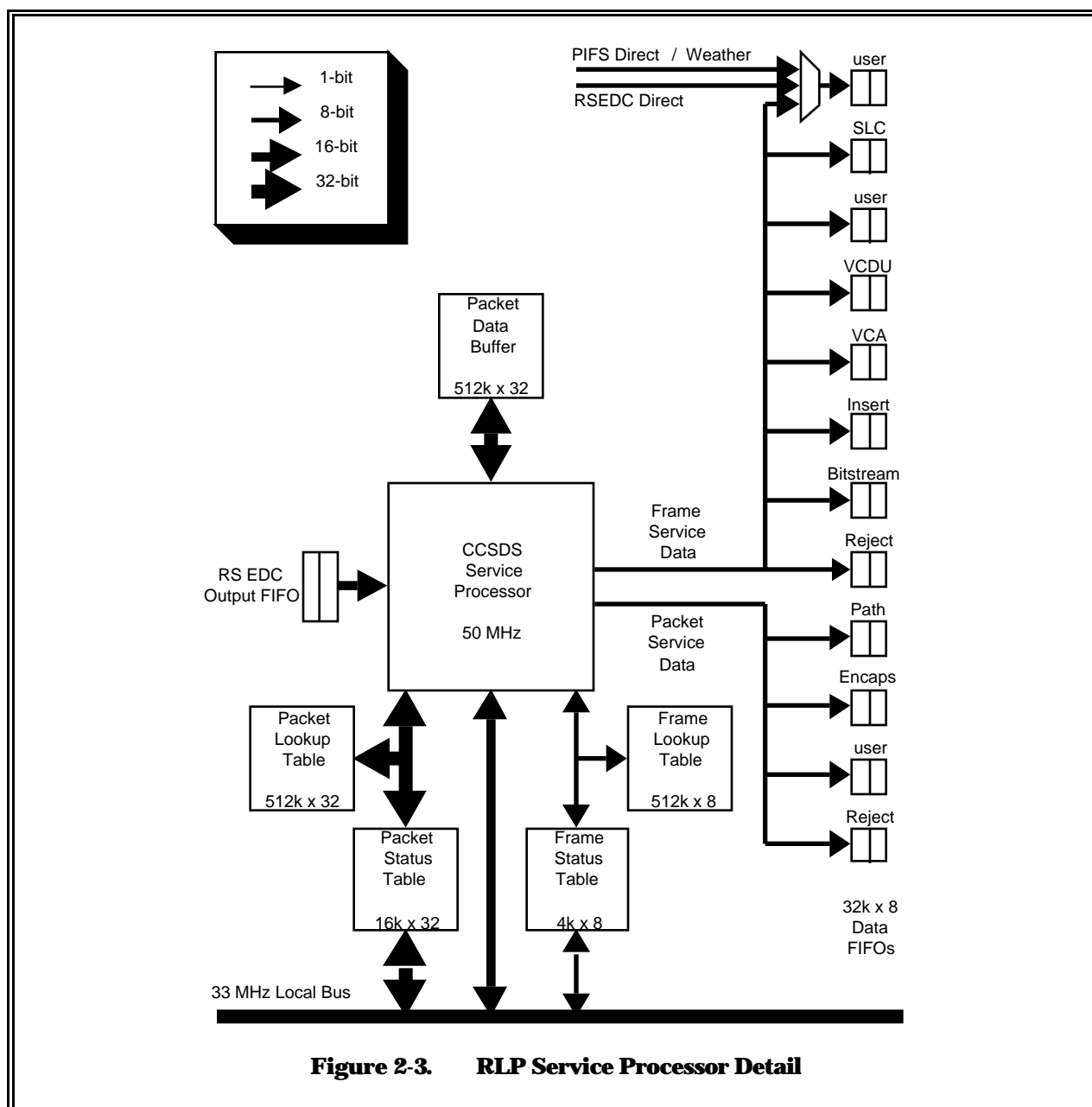
CCSDS and custom frame synchronization, non-return-to-zero (NRZ) decoding, CRC checking, BTD decoding, CCSDS day-segmented timecode generation, Weather data correlation (up to 64 bits), associated frame annotation, and programmed telemetry data input is provided by the PIFS; see Figure 2-1. It runs off the 50 MHz processing clock and is theoretically capable of operating up to 66 MHz. It can input up to one byte per clock cycle and can output up to two bytes per clock cycle through a 16-bit output port. It has thirty two 32-bit registers for control and status, and a JTAG Boundary Scan port accessible through the PCI bus; more information is available from the reference documents listed in Section 1.4.

2.4.4 REED-SOLOMON ERROR DETECTOR/CORRECTOR

Reed-Solomon error detection and correction and associated frame quality annotation is provided by the RS. It can handle up to 8 interleaves of RS(255,223)-encoded data as well as the RS(10,6)-encoded data sometimes used for CCSDS frame headers; see Figure 2-1. It runs off the 50 MHz processing clock and is theoretically capable of operating up to 66 MHz. It can input and output one or two bytes at up to one-half the processing clock frequency through 16-bit input and output ports. It has forty 16-bit registers for control and status accessible on 32-bit boundaries; more information is available from the reference documents listed in Section 1.4.

2.4.5 CCSDS SERVICE PROCESSOR

CCSDS Packet Telemetry, AOS Service Processing, and associated frame and packet annotation is provided by the SP; see Figure 2-3. It runs off the 50 MHz processing clock and is theoretically capable of operating up to 66 MHz. It can input one or two bytes at up to one-half the processing clock frequency through a 16-bit input port and can output up to two bytes per clock cycle through two 8-bit output ports: a frame service port and a packet service port. However, it can control the gating of up to 8 FIFOs on the frame service port and up to 4 FIFOs on the packet service port, for an effective maximum output of 12 bytes per clock cycle or 1800 Mbps. Typical encountered applications are expected never to employ more than 3 services on a single telemetry data byte for an SP output data rate of 450 Mbps; the RLP will not handle significantly more than that.



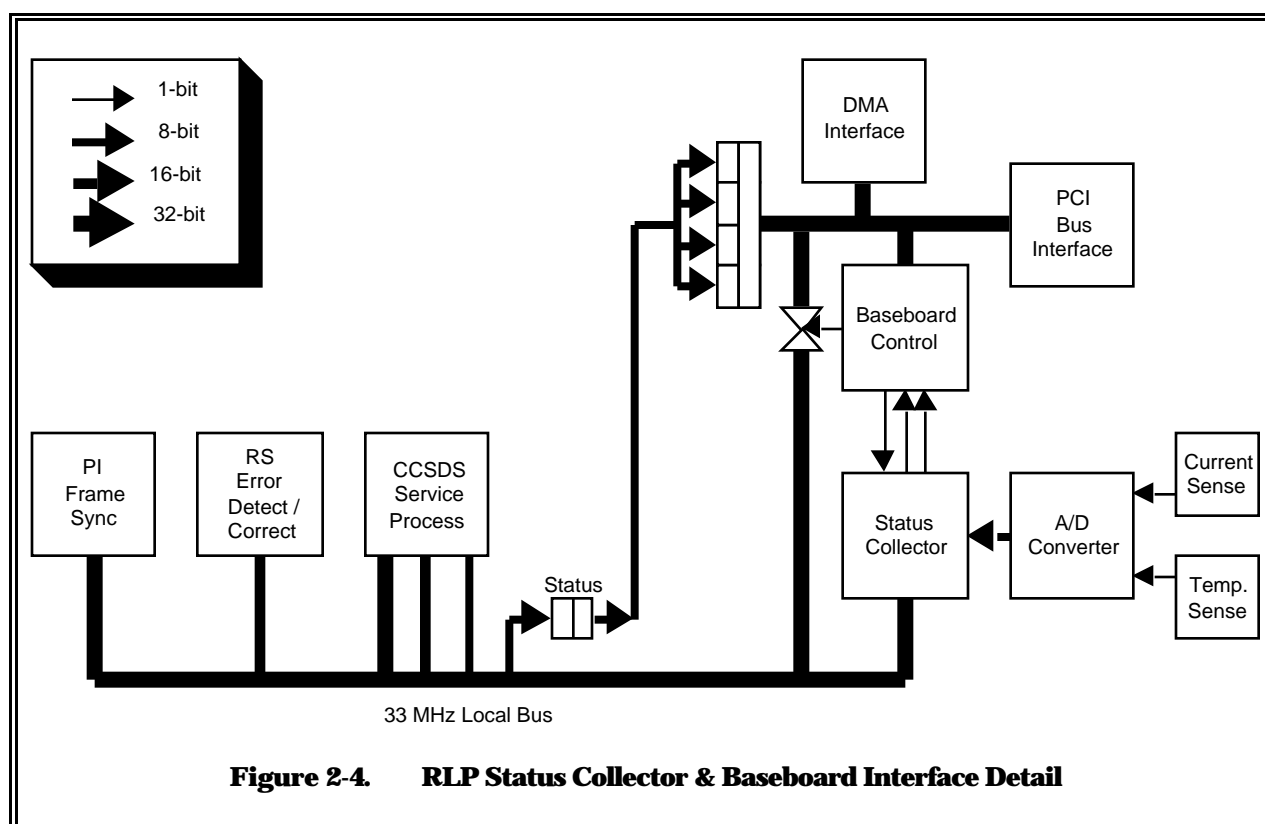
The SP is equipped with an external 512 Kbytes of memory for a frame lookup table, 2 Mbytes for a packet lookup table, 2 Mbytes for a packet data buffer, 4 Kbytes for a frame status table, and 64 Kbytes for a packet status table. The two status table memories are dual-ported for interference-free monitoring by the host. The 2 Mbytes of packet data buffer memory is not the maximum supportable by the SP, but allows 128 virtual channels with a maximum packet length of 8 Kbytes, down to 16 virtual channels with a maximum packet length of 64 Kbytes.

Each of the 12 services is provided a 32 Kbytes Data FIFO. A large FIFO is critical for buffering data while the host is involved in overhead or other non-data-transfer operations, and 32 Kbytes is the largest FIFO available in 1996.

The SP has one-hundred-and-twelve 16-bit registers for control and status accessible on 32-bit boundaries, and has a JTAG Boundary Scan port accessible through the PCI bus; more information is available from the reference documents listed in Section 1.4.

2.4.6 STATUS COLLECTOR

The Status Collector is responsible for collecting status from the other devices on the RLP and depositing it in a Data FIFO; see Figure 2-4. It is a programmable logic device running off the 33 MHz local bus clock and has two 32-bit registers. The first is the Status Request register. Writing to it first places a copy of the status request itself into the Status Data FIFO, followed by the contents of the analog-to-digital converter (ADC) register (described in Section 3.3.7.2), followed by any requested status. Requestable status consists of the 128-byte contents of the PIFS registers, the 80-byte contents of the RS registers, the 224-byte contents of the SP registers, and host-specified amounts of the SP Frame Status and Packet Status memories. When the last byte of status data has been written into the Status Data FIFO, the Status Collector asserts a signal to the Baseboard Interface (described in Section 2.4.8) that is accessible from a Baseboard Condition register and can be specified by the host to cause an interrupt.



The second Status Collector register is the ADC register. As well as being host-accessible, it is also automatically collected as part of a status request. It contains two coded values generated by the ADC: one for the total current draw on the 5V supply by the RLP, and one for the ambient temperature; see Table 3-13 for the conversion values. The ADC has eight analog inputs and constantly scans them, depositing the converted values into an internal dual-ported memory. The ADC memory is too slow to access directly from the local bus, so the Status Collector constantly scans it and places the ADC values into its own register. The ADC and Status Collector are directly connected and the scanning operation does not interfere with any other board operation.

The Status Collector also provides the ability for the host to write directly into the Status Data FIFO for flag programming and testing purposes.

The Status Collector and all points of collection operate on a segment of the local bus isolated by a switching circuit from the one over which the telemetry data flows. The switch is only closed when a host access falls within the appropriate memory region; the Baseboard Interface is responsible for arbitrating with the Status Collector for the "status bus segment" and operating the switch. The Baseboard Interface decodes the address space belonging to the different bus segments, while the Status Collector decodes the individual addresses for accessing everything on the status bus segment. Bus arbitration is discussed in detail in Section 2.4.8.

The Status Collector is in-circuit reprogrammable through a special programming / JTAG Boundary Scan port accessible through an onboard MACHPRO connector; more information is available from the reference documents listed in Section 1.4.

2.4.7 DMA INTERFACE

The DMA Interface is responsible for transferring telemetry data from the 8-bit-wide Data FIFOs to the 32k x 32-bit (128 Kbytes) Transfer FIFO and also controlling the DMA itself for efficient transfer across the PCI bus; see Figure 2-2. It is a programmable logic device running off both the 50 MHz processing clock and the 60 MHz byte-transfer clock. It can transfer data at up to 480 Mbps, more than three times a nominal 150 Mbps input data rate, and it can therefore accommodate the expected maximum SP data multiplication at this rate of three.

The DMA Interface has five 32-bit registers. The first is the DMAI Request register for channel A. The host writes the code for the desired source Data FIFO and the number of bytes to transfer. As soon as a request is written, the DMA Interface commences the flow of data at a rate of one byte per 60 MHz clock cycle. Should the Transfer FIFO become full, the DMA Interface will wait indefinitely for room to appear. If the source Data FIFO becomes empty, the DMA Interface fills the remainder of the byte transfer request with filler bytes (copies of the last valid data byte) and counts the number of filler bytes in the Fill Count register for channel A, the second register. The third and fourth registers are the DMAI Request and Fill Count registers for channel B; two identical byte transfer channels are provided to better accommodate the two DMA engines of the PBC and allow a modicum of pipelining. The channels are used in the order written; either channel may also be used exclusively. The fifth register is the DMAI Control register. It allows the host to specify the order in which bytes are packed into 32-bit words and to reset the Transfer FIFO without affecting the programmable flag settings.

The DMA Interface accommodates the emptying of the Transfer FIFO at one 32-bit word per local bus clock cycle, the same as the maximum bandwidth of the PCI bus. As part of the RLP setup, the host sets the programmable almost-empty flag of the Transfer FIFO to a value that allows reasonably large PCI bursts while not starving the Data FIFOs that are not being serviced at any particular moment. While the amount of data in the Transfer FIFO is below this specified threshold and the byte transfer from the source Data FIFO is unfulfilled, any accesses from the PBC as part of a DMA are held off, thus providing a maximally dense PCI burst. Should the DMA operation empty the Transfer FIFO before the byte transfer from the source Data FIFO is fulfilled, the DMA will once again be held off, and this bursting cycle will repeat until the byte transfer request is finally fulfilled.

At the end of a data session, any data remaining in the Data FIFOs is most efficiently flushed by transferring a block of data equal to the size of the Data FIFO and observing the Fill Count register(s) afterwards to determine how much real data existed at the beginning of the transaction.

The DMA Interface also provides the ability for the host to read the Data FIFOs directly and write the Transfer FIFO directly for flag programming and testing purposes.

The DMA Interface is in-circuit reprogrammable through a special programming / JTAG Boundary Scan port accessible through an onboard MACHPRO connector; see the documents; more information is available from the reference documents listed in Section 1.4.

2.4.8 BASEBOARD INTERFACE

The Baseboard Interface is responsible for a number of functions, the most complex of which is controlling and generating the interrupt from some 54 possible sources; see Figure 2-4. It also provides miscellaneous board control functions, decodes the major address spaces and the DMA-Chaining Memory space, and arbitrates with the Status Collector for the status bus segment. It is a programmable logic device running off the 33 MHz local bus clock and has five 32-bit registers. The first is the Baseboard Control register, which allows the host to toggle the LED, select single-ended or differential ECL data input, select onboard or off-board PIFS timecode reference clock, select SP/FS-Direct/RS-Direct telemetry data routing, put all programmable FIFOs in flag-programming mode, select whether the Interrupt Condition registers should indicate the raw or the masked form of the interrupt sources, reset 16 separate groups of devices, and disable the interrupt feature. The second and third registers are the Interrupt Source Mask registers, and the fourth and fifth are the Interrupt Source Condition registers. 54 possible interrupting signals are distributed among the two registers in each pair: as many FIFO flag signals as permitted by available pins on the device; the foremost FIFO overflow condition, which is detected, latched, and made clearable; the Status Collection Fulfilled signal; and the Optional Sorting Module interrupt signal. The latter also goes directly to the PBC, so either (or both) may be disabled (masked) if desired.

To perform the function of arbitrating for the status bus segment when the Baseboard Interface detects an address to status bus space, it asserts a Status Bus Request signal that goes directly into the Status Collector. When the Status Collector is ready to give up its bus, if necessary pausing a Status Collection Operation, it asserts a Status Bus Grant signal that goes directly back to the Baseboard Interface. The Baseboard Interface then closes the bus switch, connecting the two segments of the local bus together, and the status bus space access proceeds with the Status Collector decoding the specific address. The Status Collector continues granting the status bus long enough so that an immediately subsequent status bus space access avoids re-arbitration. Eventually after the Baseboard Interface deasserts the Status Bus Request signal and the immediately subsequent access does not require the status bus, the Status Collector deasserts the Status Bus Grant signal and, if necessary, continues a Status Collection Operation.

The Baseboard Interface is in-circuit reprogrammable through a special programming / JTAG Boundary Scan port accessible through an onboard MACHPRO connector; more information is available from the reference documents listed in Section 1.4.

2.5 INTERRUPTS

A single PCI device such as the RLP is allowed by the PCI Specification to generate only a single interrupt. The source of the interrupt will be one or more of the 54 conditions indicated in the Baseboard Condition registers shown in Section 3.3.2. If an Optional Sorting Module is present, the PBC may be set up to generate the interrupt directly from it; more information is available from the reference documents listed in Section 1.4.

SECTION 3 MEMORY MAP AND REGISTER DEFINITIONS

3.1 INTRODUCTION

The RLP essentially consists of four ASICs, three CPLDs, and a number of memories which together require a total of 14 address spaces. These are arranged within a single 2 Mbytes of PCI address space. If an Optional Sorting Module is present, its address space begins immediately following the 2 Mbytes baseboard space, and the amount is defined by the Module as indicated in its hardware definition document. Only 32-bit accesses on 32-bit boundaries are supported.

This section provides the memory map and register definitions for the RLP.

3.2 MEMORY MAP

Table 3-1 lists the overall address map of the RLP.

Table 3-1. RLP Memory Map

Address (Hex)	Device/Function
00 0000 - 01 FFFF	DMA-Chaining/User Memory
02 0000 - 03 FFFF	unused (aliased DMA-Chaining/User Memory)
04 0000 - 04 0013	Baseboard Interface Registers
04 0014 - 04 001F	unused (returns 0)
04 0020 - 07 FFFF	unused (aliased Baseboard Interface Registers)
08 0000 - 09 FFFF	DMA Interface Transfer FIFO Output Space
0A 0000 - 0A 0013	DMA Interface Registers
0A 0014 - 0A 001F	unused (returns 0)
0A 0020 - 0B FFFF	unused (aliased DMA Interface Registers)
0C 0000 - 0D FFFF	Data FIFO Output Space
0E 0000 - 0F FFFF	DMA Interface Transfer FIFO Input Space
10 0000 - 10 000B	Status Collector Registers
10 000C - 10 001F	unused (returns 0)
10 0020 - 10 7FFF	unused (aliased Status Collector Registers)
10 8000 - 10 FFFF	Status FIFO Input Space
11 0000 - 11 007F	PIFS Registers
11 0080 - 11 FFFF	unused (aliased PIFS Registers)
12 0000 - 12 009F	RS Registers
12 00A0 - 12 00FF	unused (returns 0)
12 0100 - 12 FFFF	unused (aliased RS Registers)
13 0000 - 13 FFFF	RS Internal Data Memory
14 0000 - 14 01BF	SP Registers
14 01C0 - 14 01FF	unused (returns 0)
14 0200 - 14 FFFF	unused (aliased SP Registers)
15 0000 - 15 3FFF	SP Frame Status Memory
15 4000 - 15 FFFF	unused (aliased SP Frame Status Memory)
16 0000 - 16 FFFF	SP Packet Status Memory
17 0000 - 17 FFFF	unused (aliased SP Packet Status Memory)
18 0000 - 1B FFFF	SP Internal & External Memories
1C 0000 - 1F FFFF	unused
20 0000 - ...	OSM (if present)

3.3 **MEMORY MAP DESCRIPTION AND REGISTER DEFINITIONS**

3.3.1 **DMA-CHAINING/USER MEMORY \$00 0000**

The PBC has a DMA chaining feature which requires a small amount of random-access memory (RAM) to reside on the local bus; more information is available from the reference documents listed in Section 1.4. 128 Kbytes of 32-bit-wide fast static RAM (SRAM) has been provided to support this feature. It resides at address \$00 0000 through \$01 FFFF. This is more than DMA chaining requires; any surplus is available to the host for general use.

3.3.2 **BASEBOARD INTERFACE REGISTERS \$04 0000**

The Baseboard Interface contains five contiguous 32-bit-wide registers beginning at address \$04 0000 for setup, control, and monitoring RLP functions that do not belong to any of the other RLP subsystems. Accessing these registers during a DMA operation will severely degrade performance and is strongly discouraged; this is the only such limitation.

3.3.2.1 **Control Register \$04 0000**

The Baseboard Interface Control register resides at address \$04 0000 and is both readable and writeable, and contains 24 bits for controlling all of the RLP baseboard functions except the interrupt sources. See Table 3-2.

Table 3-2. Baseboard Interface Control Register \$04 0000

Bit(s)	Function	Default
<31-25>	unused	0
<24>	Interrupt Source Indication Mode (0=Raw, 1=Masked Shown Deasserted)	0
<23>	unused	0
<22>	Reset OSM (0=Reset, 1=Normal)	0
<21>	Reset DMA Interface Transfer FIFOs (Master) (0=Reset, 1=Normal)	0
<20>	Reset DMA Interface (0=Reset, 1=Normal)	0
<19>	Reset Status Collector Data FIFO (Partial) (0=Reset, 1=Normal)	0
<18>	Reset Status Collector Data FIFO (Master) (0=Reset, 1=Normal)	0
<17>	Reset Status Collector (0=Reset, 1=Normal)	0
<16>	Reset SP Packet Service Data FIFOs (Partial) (0=Reset, 1=Normal)	0
<15>	Reset SP Packet Service Data FIFOs (Master) (0=Reset, 1=Normal)	0
<14>	Reset SP Frame Service Data FIFOs (Partial) (0=Reset, 1=Normal)	0
<13>	Reset SP Frame Service Data FIFOs (Master) (0=Reset, 1=Normal)	0
<12>	Reset SP (0=Reset, 1=Normal)	0
<11>	Reset RS-to-SP FIFO (0=Reset, 1=Normal)	0
<10>	Reset RS (0=Reset, 1=Normal)	0
<09>	Reset PIFS-to-RS FIFO (0=Reset, 1=Normal)	0
<08>	Reset PIFS (0=Reset, 1=Normal)	0
<07>	Reset ECL Serial-to-Parallel Converter (1=Reset, 0=Normal)	1
<06>	Universal FIFO Programmable Flag Load (0=Load, 1=Normal)	0
<05>	PIFS Timecode Reference Clock Select (0=Internal, 1=External)	0
<04-03>	Processing Mode (00=SP, 01=FS-Direct, 10=SP, 11=RS-Direct)	0
<02>	ECL Data Input Type Select (0=Single-Ended, 1=Differential)	1
<01>	LED Control (0=On, 1=Off)	0
<00>	Clear Foremost FIFO Overflow Condition (1=Clear, 0=Normal)	1

3.3.2.2 Interrupt Source Mask Register A \$04 0004

The Baseboard Interface Interrupt Source Mask register A resides at address \$04 0004 and is both readable and writeable. It contains 30 bits for controlling the masking of one half of the RLP baseboard interrupt source signals and one bit for disabling the entire interrupt. See Table 3-3.

Table 3-3. Baseboard Interface Interrupt Source Mask Register A \$04 0004

Bit(s)	Function	Default
<31>	Enable Baseboard Interrupt (1=Enable, 0=Disable)	0
<30>	Mask Foremost FIFO Overflow Condition (0=Mask, 1=Use)	0
<29>	Mask SP Status Data FIFO Almost-Empty Flag (0=Mask, 1=Use)	0
<28>	Mask SP Frame Reject FIFO Almost-Empty Flag (0=Mask, 1=Use)	0
<27>	Mask SP Bitstream Service FIFO Almost-Empty Flag (0=Mask, 1=Use)	0
<26>	Mask SP Frame Service 3 / FS-Direct / RS-Direct FIFO Almost-Empty Flag (0=Mask, 1=Use)	0
<25>	Mask SP Frame Service 2 FIFO Almost-Empty Flag (0=Mask, 1=Use)	0
<24>	Mask SP Frame Service 1 FIFO Almost-Empty Flag (0=Mask, 1=Use)	0
<23>	Mask SP Static Service 3 FIFO Almost-Empty Flag (0=Mask, 1=Use)	0
<22>	Mask SP Static Service 2 FIFO Almost-Empty Flag (0=Mask, 1=Use)	0
<21>	Mask SP Static Service 1 FIFO Almost-Empty Flag (0=Mask, 1=Use)	0
<20>	Mask SP Packet Reject FIFO Almost-Empty Flag (0=Mask, 1=Use)	0
<19>	Mask SP Packet Service 3 FIFO Almost-Empty Flag (0=Mask, 1=Use)	0
<18>	Mask SP Packet Service 2 FIFO Almost-Empty Flag (0=Mask, 1=Use)	0
<17>	Mask SP Packet Service 1 FIFO Almost-Empty Flag (0=Mask, 1=Use)	0
<16>	Mask SP Status FIFO Almost-Full Flag (0=Mask, 1=Use)	0
<15>	Mask SP Frame Reject FIFO Almost-Full Flag (0=Mask, 1=Use)	0
<14>	Mask SP Bitstream Service FIFO Almost-Full Flag (0=Mask, 1=Use)	0
<13>	Mask SP Frame Service 3 / FS-Direct / RS-Direct FIFO Almost-Full Flag (0=Mask, 1=Use)	0
<12>	Mask SP Frame Service 2 FIFO Almost-Full Flag (0=Mask, 1=Use)	0
<11>	Mask SP Frame Service 1 FIFO Almost-Full Flag (0=Mask, 1=Use)	0
<10>	Mask SP Static Service 3 FIFO Almost-Full Flag (0=Mask, 1=Use)	0
<09>	Mask SP Static Service 2 FIFO Almost-Full Flag (0=Mask, 1=Use)	0
<08>	Mask SP Static Service 1 FIFO Almost-Full Flag (0=Mask, 1=Use)	0
<07>	Mask SP Packet Reject FIFO Almost-Full Flag (0=Mask, 1=Use)	0
<06>	Mask SP Packet Service 3 FIFO Almost-Full Flag (0=Mask, 1=Use)	0
<05>	Mask SP Packet Service 2 FIFO Almost-Full Flag (0=Mask, 1=Use)	0
<04>	Mask SP Packet Service 1 FIFO Almost-Full Flag (0=Mask, 1=Use)	0
<03>	Mask DMAI FIFO Full Flag (0=Mask, 1=Use)	0
<02>	Mask RS-to-SP FIFO Full Flag (0=Mask, 1=Use)	0
<01>	Mask PIFS-to-RS FIFO Full Flag (0=Mask, 1=Use)	0
<00>	Mask OSM Interrupt Signal (0=Mask, 1=Use)	0

3.3.2.3 Interrupt Source Mask Register B \$04 0008

The Baseboard Interface Interrupt Source Mask register B resides at address \$04 0008 and is both readable and writeable. It contains 23 bits for controlling the masking of the other half of the RLP baseboard interrupt source signals. See Table 3-4.

Table 3-4. Baseboard Interface Interrupt Source Mask Register B \$04 0008

Bit(s)	Function	Default
--------	----------	---------

<31-23>	unused	0
<22>	Mask Status Collection Done Signal (0=Mask, 1=Use)	0
<21>	Mask SP Status FIFO Empty Flag (0=Mask, 1=Use)	0
<20>	Mask SP Frame Reject FIFO Empty Flag (0=Mask, 1=Use)	0
<19>	Mask SP Bitstream Service FIFO Empty Flag (0=Mask, 1=Use)	0
<18>	Mask SP Frame Service 3 / FS-Direct / RS-Direct FIFO Empty Flag (0=Mask, 1=Use)	0
<17>	Mask SP Frame Service 2 FIFO Empty Flag (0=Mask, 1=Use)	0
<16>	Mask SP Frame Service 1 FIFO Empty Flag (0=Mask, 1=Use)	0
<15>	Mask SP Static Service 3 FIFO Empty Flag (0=Mask, 1=Use)	0
<14>	Mask SP Static Service 2 FIFO Empty Flag (0=Mask, 1=Use)	0
<13>	Mask SP Static Service 1 FIFO Empty Flag (0=Mask, 1=Use)	0
<12>	Mask SP Packet Reject FIFO Empty Flag (0=Mask, 1=Use)	0
<11>	Mask SP Packet Service 3 FIFO Empty Flag (0=Mask, 1=Use)	0
<10>	Mask SP Packet Service 2 FIFO Empty Flag (0=Mask, 1=Use)	0
<09>	Mask SP Packet Service 1 FIFO Empty Flag (0=Mask, 1=Use)	0
<08>	Mask DMAI FIFO Empty Flag (0=Mask, 1=Use)	0
<07>	Mask RS-to-SP FIFO Empty Flag (0=Mask, 1=Use)	0
<06>	Mask PIFS-to-RS FIFO Empty Flag (0=Mask, 1=Use)	0
<05>	Mask DMAI FIFO Half-Full Flag (0=Mask, 1=Use)	0
<04>	Mask RS-to-SP FIFO Half-Full Flag (0=Mask, 1=Use)	0
<03>	Mask PIFS-to-RS FIFO Half-Full Flag (0=Mask, 1=Use)	0
<02>	Mask DMAI FIFO Almost-Full Flag (0=Mask, 1=Use)	0
<01>	Mask RS-to-SP FIFO Almost-Full Flag (0=Mask, 1=Use)	0
<00>	Mask PIFS-to-RS FIFO Almost-Full Flag (0=Mask, 1=Use)	0

3.3.2.4 Interrupt Source Register A \$04 000C

The Baseboard Interface Interrupt Source register A resides at address \$04 000C and is only readable; writes have no effect. It contains 31 bits indicating either the raw or masked states of one half of the RLP baseboard interrupt source signals, depending on how bit 24 of the Baseboard Interface Control register \$04 0000 is set. See Table 3-5.

Table 3-5. Baseboard Interface Interrupt Source Register A \$04 000C

Bit(s)	Function
<31>	0 (unused)
<30>	Overflow Condition (1=Overflow Int)
<29>	SP Status FIFO Almost-Empty (1=Not Almost-Empty Int)
<28>	SP Frame Reject FIFO Almost-Empty (1=Not Almost-Empty Int)
<27>	SP Bitstream Service FIFO Almost-Empty (1=Not Almost-Empty Int)
<26>	SP Frame Service 3 / FS-Direct / RS-Direct FIFO Almost-Empty (1=Not Almost-Empty Int)
<25>	SP Frame Service 2 FIFO Almost-Empty (1=Not Almost-Empty Int)
<24>	SP Frame Service 1 FIFO Almost-Empty (1=Not Almost-Empty Int)
<23>	SP Static Service 3 FIFO Almost-Empty (1=Not Almost-Empty Int)
<22>	SP Static Service 2 FIFO Almost-Empty (1=Not Almost-Empty Int)
<21>	SP Static Service 1 FIFO Almost-Empty (1=Not Almost-Empty Int)
<20>	SP Packet Reject FIFO Almost-Empty (1=Not Almost-Empty Int)
<19>	SP Packet Service 3 FIFO Almost-Empty (1=Not Almost-Empty Int)
<18>	SP Packet Service 2 FIFO Almost-Empty (1=Not Almost-Empty Int)
<17>	SP Packet Service 1 FIFO Almost-Empty (1=Not Almost-Empty Int)
<16>	SP Status FIFO Almost-Full (0=Almost-Full Int)

<15>	SP Frame Reject FIFO Almost-Full (0=Almost-Full Int)
<14>	SP Bitstream Service FIFO Almost-Full (0=Almost-Full Int)
<13>	SP Frame Service 3 / FS-Direct / RS-Direct FIFO Almost-Full (0=Almost-Full Int)
<12>	SP Frame Service 2 FIFO Almost-Full (0=Almost-Full Int)
<11>	SP Frame Service 1 FIFO Almost-Full (0=Almost-Full Int)
<10>	SP Static Service 3 FIFO Almost-Full (0=Almost-Full Int)
<09>	SP Static Service 2 FIFO Almost-Full (0=Almost-Full Int)
<08>	SP Static Service 1 FIFO Almost-Full (0=Almost-Full Int)
<07>	SP Packet Reject FIFO Almost-Full (0=Almost-Full Int)
<06>	SP Packet Service 3 FIFO Almost-Full (0=Almost-Full Int)
<05>	SP Packet Service 2 FIFO Almost-Full (0=Almost-Full Int)
<04>	SP Packet Service 1 FIFO Almost-Full (0=Almost-Full Int)
<03>	DMAI FIFO Full (0=Full Int)
<02>	RS-to-SP FIFO Full (0=Full Int)
<01>	PIFS-to-RS FIFO Full (0=Full Int)
<00>	OSM Interrupt Signal (0=OSM Int)

3.3.2.5 Interrupt Source Register B \$04 0010

The Baseboard Interface Interrupt Source register B resides at address \$04 0010 and is only readable; writes have no effect. It contains 23 bits indicating either the raw or masked states of the other half of the RLP baseboard interrupt source signals, depending on how bit 24 of the Baseboard Interface Control register \$04 0000 is set, and one bit indicating the presence of an Optional Sorting Module which does not cause an interrupt and is not maskable. See Table 3-6.

Table 3-6. Baseboard Interface Interrupt Source Register B \$04 0010

Bit(s)	Function
<31-24>	0 (unused)
<23>	OSM Present (1=Present, 0=Absent)
<22>	Staus Collection Done Signal (1=Done Int)
<21>	SP Status FIFO Empty (1=Not Empty Int)
<20>	SP Frame Reject FIFO Empty (1=Not Empty Int)
<19>	SP Bitstream Service FIFO Empty (1=Not Empty Int)
<18>	SP Frame Service 3 / FS-Direct / RS-Direct FIFO Empty (1=Not Empty Int)
<17>	SP Frame Service 2 FIFO Empty (1=Not Empty Int)
<16>	SP Frame Service 1 FIFO Empty (1=Not Empty Int)
<15>	SP Static Service 3 FIFO Empty (1=Not Empty Int)
<14>	SP Static Service 2 FIFO Empty (1=Not Empty Int)
<13>	SP Static Service 1 FIFO Empty (1=Not Empty Int)
<12>	SP Packet Reject FIFO Empty (1=Not Empty Int)
<11>	SP Packet Service 3 FIFO Empty (1=Not Empty Int)
<10>	SP Packet Service 2 FIFO Empty (1=Not Empty Int)
<09>	SP Packet Service 1 FIFO Empty (1=Not Empty Int)
<08>	DMAI FIFO Empty (1=Not Empty Int)
<07>	RS-to-SP FIFO Empty (1=Not Empty Int)
<06>	PIFS-to-RS FIFO Empty (1=Not Empty Int)
<05>	DMAI FIFO Half-Full (0=Half-Full Int)
<04>	RS-to-SP FIFO Half-Full (0=Half-Full Int)
<03>	PIFS-to-RS FIFO Half-Full (0=Half-Full Int)
<02>	DMAI FIFO Almost-Full (0=Almost-Full Int)
<01>	RS-to-SP FIFO Almost-Full (0=Almost-Full Int)

<00>	PIFS-to-RS FIFO Almost-Full (0=Almost-Full Int)
------	---

3.3.3 DMA INTERFACE TRANSFER FIFO OUTPUT SPACE \$08 0000

The DMA Interface Transfer FIFO holds up to 128 Kbytes of telemetry data, annotation data, and board subsystem status data from the 8-bit-wide Data FIFOs packed into 32-bit words for eventual transfer across the PCI bus. The 32-bit-wide FIFO output is aliased over 128 Kbytes of address space from \$08 0000 through \$09 FFFF so that sequential reads from sequential addresses perform sequential reads from the FIFO output up to the 128 Kbytes depth of the FIFO. The Transfer FIFO also has programmable almost-full and almost-empty flags which are used by the DMAI Controller, and their corresponding offset registers can be read through this space. It is readable only; writes have no effect.

3.3.4 DMA INTERFACE CONTROLLER \$0A 0000

The DMAI Controller contains five contiguous 32-bit-wide registers beginning at address \$0A 0000 for setting up and initiating up to two sequential Data Transfers from the 8-bit-wide Data FIFOs into the 32-bit-wide DMAI Transfer FIFO. Accessing these registers during a DMA operation will severely degrade performance and is strongly discouraged; some registers have further limitations as noted.

3.3.4.1 Channel A Control Register \$0A 0000

The DMAI Channel A Control register resides at address \$0A 0000 and is readable and writeable. It contains 17 bits for specifying the number of data bytes to be transferred into the DMAI Transfer FIFO, and four bits to select which of the 13 Data FIFOs (four SP Packet Services, eight SP Frame Services, and one Status) to obtain the data from. The act of writing this register selects the source Data FIFO and initiates a Data Transfer Operation. The source Data FIFO selection applies to both the Data Transfer and to reads from the Data FIFO Output Space described in Section 3.3.5; if only Data FIFO selection without Data Transfer is required, a Data Transfer Size of zero is recommended. If this register is read during a Channel A Data Transfer Operation, the Data Transfer Size field will indicate the number of bytes remaining to be transferred before completion. Writing this register during a Channel A Data Transfer Operation will cause unpredictable results and must not be allowed to occur. See Table 3-7.

Table 3-7. DMA Interface Channel A Control Register \$0A 0000

Bit(s)	Function	Default
<31-21>	unused	0
<20-17>	Source Data FIFO Selection: 0000 = SP Packet Service 1 0001 = SP Packet Service 2 0010 = SP Packet Service 3 0011 = SP Packet Reject Service 0100 = SP Frame Service 1 0101 = SP Frame Service 2 0110 = SP Frame Service 3 or RS-Direct or FS-Direct 0111 = SP Static Service 1 1000 = SP Static Service 2 1001 = SP Static Service 3 1010 = SP Bitstream Service 1011 = SP Frame Reject Service 1100 = Status 1101, 1110, 1111 = no FIFO selected	0
<16-00>	Channel A Data Transfer Size (Remaining), in Bytes	0

3.3.4.2 Channel B Control Register \$0A 0004

The DMAI Channel B Control register resides at address \$0A 0004 and is readable and writeable. It contains 17 bits for specifying the number of data bytes to be transferred into the DMA Interface Transfer FIFO, and four bits to select which of the 13 Data FIFOs to obtain the data from. The act of writing this register selects the source Data FIFO and initiates a Data Transfer Operation. The source Data FIFO selection applies to both the Data Transfer and to reads from the Data FIFO Output Space described in Section 3.3.5; if only Data FIFO selection without Data Transfer is required, a Data Transfer Size of zero is recommended. If this register is read during a Channel B Data Transfer Operation, the Data Transfer Size field will indicate the number of bytes remaining to be transferred before completion. Writing this register during a Channel B Data Transfer Operation will cause unpredictable results and must not be allowed to occur. See Table 3-8.

Table 3-8. DMA Interface Channel B Control Register \$0A 0004

Bit(s)	Function	Default
<31-21>	unused	0
<20-17>	Source Data FIFO Selection: 0000 = SP Packet Service 1 0001 = SP Packet Service 2 0010 = SP Packet Service 3 0011 = SP Packet Reject Service 0100 = SP Frame Service 1 0101 = SP Frame Service 2 0110 = SP Frame Service 3 or RS-Direct or FS-Direct 0111 = SP Static Service 1 1000 = SP Static Service 2 1001 = SP Static Service 3 1010 = SP Bitstream Service 1011 = SP Frame Reject Service 1100 = Status 1101, 1110, 1111 = no FIFO selected	0
<16-00>	Channel B Data Transfer Size (Remaining), in Bytes	0

3.3.4.3 Channel A Fill Count Register \$0A 0008

The DMAI Channel A Fill Count register resides at address \$0A 0008 and is readable only; writes have no effect. It contains 17 bits that indicate the number of filler bytes (copies of the last valid data byte) that have been appended to the most recent Channel A Data Transfer. Filling will occur when the specified Data FIFO contains less data than specified for the Transfer Size. During a data session, this situation is easily avoidable and should not be encountered, but when a data session is concluded it can be used to indicate the amount of data that remained in a Data FIFO if it was flushed using a maximum-sized Data Transfer. See Table 3-9.

Table 3-9. DMA Interface Channel A Fill Count Register \$0A 0008

Bit(s)	Function
<31-17>	0 (unused)
<16-00>	Count of Fill Bytes in Most Recent Channel A Data Transfer

3.3.4.4 Channel B Fill Count Register \$0A 000C

The DMAI Channel B Fill Count register resides at address \$0A 000C and is readable only; writes have no effect. It contains 17 bits that indicate the number of filler bytes (copies of the last valid data byte) that have been appended to the most recent Channel B Data Transfer. Filling will occur when the specified Data FIFO contains less data than specified for the Transfer Size. During a data session, this situation is easily avoidable and should not be encountered, but when a data session is concluded it can be used to indicate the amount of data that remained in a Data FIFO if it was flushed using a maximum-sized Data Transfer. See Table 3-10.

Table 3-10. DMA Interface Channel B Fill Count Register \$0A 000C

Bit(s)	Function
<31-17>	0 (unused)
<16-00>	Count of Fill Bytes in Most Recent Channel B Data Transfer

3.3.4.5 General Control Register \$0A 0010

The DMAI General Control register resides at address \$0A 0010 and is readable and writeable. It contains two bits. One bit performs a partial reset of the DMAI Transfer FIFO whereby the data pointers are reset but the programmable-flag values are maintained. The other bit selects the Data Byte Ordering Mode whereby the chronologically-sequential data bytes are packed into 32-bit words ordered first byte to either bits 31-24 or 7-0, second byte to bits 23-16 or 15-8, third byte to bits 15-8 or 23-16, and fourth byte to bits 0-7 or 31-24. Writing this register during a Data Transfer Operation will cause unpredictable results and must not be allowed to occur. See Table 3-11.

Table 3-11. DMA Interface General Control Register \$0A 0010

Bit(s)	Function	Default
<31-02>	unused	0
<01>	Partial Reset of DMA Interface Transfer FIFO (0=Reset, 1=Normal)	1
<00>	Data Byte Order Mode: 0 = first byte to D<31-24> 1 = first byte to D<07-00>	0

3.3.5 DMA INTERFACE DATA FIFO OUTPUT SPACE \$0C 0000

Each 8-bit-wide Data FIFO holds up to 32 Kbytes of telemetry data, annotation data, or board subsystem status data from the Status Collector, SP, RS, or PIFS for eventual transfer into the 32-bit-wide DMAI Transfer FIFO. For testing purposes, the 8-bit-wide outputs (actually 9 bits including end-of-data marker) of these Data FIFOs can be read directly through the DMAI Data FIFO Output Space. One Data FIFO output is connected to bits 8-0 of the 32-bit access bus and aliased over 128 Kbytes of address space from \$0C 0000 through \$0D FFFF so that sequential reads from sequential addresses perform sequential reads from the FIFO up to its 32k depth. The Data FIFOs also have programmable almost-full and almost-empty flags which are used by the Baseboard Interface, Status Collector, and SP, and their corresponding offset registers can be read

through this space. The specific source Data FIFO must be selected beforehand using either the DMAI Channel A or B Control registers \$0A 0000 or \$0A 0004. If no Data Transfer Operation is desired, using a Data Transfer Size of zero is recommended. If no Data FIFO has been selected, either explicitly or implicitly by the last Data Transfer Operation, reads from the Data FIFO Output Space will return unpredictable values but will have no other effect. The space is readable only; writes will have no effect.

3.3.6 DMA INTERFACE TRANSFER FIFO INPUT SPACE \$0E 0000

The DMAI Transfer FIFO holds up to 128 Kbytes of telemetry data, annotation data, and board subsystem status data from the 8-bit-wide Data FIFOs packed into 32-bit words for eventual transfer across the PCI bus. For testing purposes, the 32-bit-wide FIFO input is aliased over 128 Kbytes of address space from \$0E 0000 through \$0F FFFF so that sequential writes to sequential addresses perform sequential writes to the FIFO input up to the 128 Kbytes depth of the FIFO. The 32-bit-wide FIFO is composed of four 8-bit-wide FIFOs in parallel, and for each 32-bit write to the space, the least-significant 8-bits are written to all four FIFOs simultaneously. The Transfer FIFO also has programmable almost-full and almost-empty flags which are used by the DMAI Controller, and their corresponding offset registers can be written through this space. It is writeable only; reads will have no effect but will return unpredictable values.

3.3.7 STATUS COLLECTOR \$10 0000

The Status Collector contains two contiguous 32-bit-wide registers beginning at address \$10 0000 for manipulating status information and testing. Accessing these registers during a DMA operation will severely degrade performance and is strongly discouraged; some registers have further limitations as noted.

3.3.7.1 Status Request Register \$10 0000

The Status Collector Status Request register resides at address \$10 0000 and is readable and writeable. It contains 27 bits for specifying the types and quantities of status data to collect and deposit into the 8-bit-wide Status Data FIFO. The act of writing this register initiates a Status Collection Operation. The total number of status data bytes that a given status request will deposit in the Status Data FIFO equals the sum of the requested data plus 8, the size of the status data header which consists of the contents of the Status Request register \$10 0000 followed by the contents of the Analog Values register \$10 0004. The Status Collector does not put the "spacer" bytes used to align 8- or 16-bit data to 32-bit boundaries into the Status Data FIFO, so the structure of the status data differs from that which is directly read and written. Writing this register during a Status Collection Operation will cause unpredictable results and must not be allowed to occur. See Table 3-12.

Table 3-12. Status Collector Status Request Register \$10 0000

Bit(s)	Function	Default
<31-27>	unused	0
<26-13>	Number of 4-Byte SP Packet Status Half-Records to Collect - 1	0
<12>	0 = No Effect 1 = Collect SP Packet Status Half-Records...	0
<11-04>	Number of 16-Byte SP Frame Status Half-Records to Collect - 1	0
<03>	0 = No Effect 1 = Collect SP Frame Status Half-Records...	0
<02>	0 = No Effect 1 = Collect SP Registers (112 W, 224 B)	0
<01>	0 = No Effect 1 = Collect RS Registers (40 W, 80 B)	0
<00>	0 = No Effect 1 = Collect PIFS Registers (32 LW, 128 B)	0

3.3.7.2 Analog Values Register \$10 0004

The Status Collector Analog Values register resides at address \$10 0004 and is only readable. It contains 16 bits indicating the digital values of various measured analog quantities. It is updated constantly and automatically. Writing this register has no effect. See Table 3-13.

Table 3-13. Status Collector Analog Values Register \$10 0004

Bit(s)	Function
<31-16>	0 (unused)
<15-08>	Ambient Board Temperature Code: x 2.03 = degrees Celsius
<07-00>	5 Volt Board Power Code: x .203 = Watts, or x .0406 = Amperes

3.3.8 STATUS DATA FIFO INPUT SPACE \$10 8000

The 9-bit-wide Status Data FIFO holds up to 32 Kbytes of status data and end-of-data markers from the Status Collector for eventual transfer into the 32-bit-wide DMAI Transfer FIFO. For testing purposes, the 8-bit Status Data FIFO input (actually 9 bits including end-of-data marker) is aliased over 128 Kbytes of address space from \$10 8000 through \$10 FFFF so that sequential writes to sequential addresses perform sequential writes to the FIFO up to its 32 Kbytes depth. For each 32-bit write to the space, only the least-significant 9-bits are written into the FIFO. The Status Data FIFO also has programmable almost-full and almost-empty flags which are used by the Status Collector, and its corresponding offset registers can be written through this space. It is writeable only; reading this space has no effect but will return unpredictable values.

3.3.9 PIFS REGISTER SPACE \$11 0000

The PIFS contains 32 contiguous 32-bit-wide registers beginning at address \$11 0000 for setup, control, monitoring, data injection, and testing. Access of these registers during a DMA operation should only be performed by the Status Collector for a Status Collection Operation; host access will severely degrade performance and is strongly discouraged. Some registers have further limitations as noted.

[PIFS Register Descriptions to be copied here from PIFS Document by Technical Publications]

3.3.10 RS REGISTER SPACE \$12 0000

The RS contains forty 16-bit-wide registers spread out into 40 contiguous 32-bit-wide locations one 16-bit register per 32-bit word beginning at address \$12 0000 for setup, control, monitoring, data injection, and testing. Access of these registers during a DMA operation should only be performed by the Status Collector for a Status Collection Operation; host access will severely degrade performance and is strongly discouraged. Some registers have further limitations as noted.

[RS Register Descriptions to be copied here from RS Document by Technical Publications]

3.3.11 RS MEMORY SPACE \$13 0000

The RS contains 16 Kbytes of 8-bit-wide internal RAM spread out into 64 Kbytes of 32-bit-wide address space one memory byte per 32-bit access word beginning at address \$13 0000 for working storage. This memory is not accessible via Status Collection; host access will severely degrade performance and is strongly discouraged. Writing it while the RS is running will have unpredictable results and must not be allowed to occur.

[RS Memory Description to be copied here from RS Document by Technical Publications]

3.3.12 SP REGISTER SPACE \$14 0000

The SP contains one-hundred-and-twelve 16-bit-wide registers spread out into 112 contiguous 32-bit-wide locations one 16-bit register per 32-bit word beginning at address \$14 0000 for setup, control, monitoring, data injection, and testing. Access of these registers during a DMA operation should only be performed by the Status Collector for a Status Collection Operation; host access will severely degrade performance and is strongly discouraged. Some registers have further limitations as noted.

[SP Register Descriptions to be copied here from SP Document by Technical Publications]

3.3.13 SP FRAME STATUS MEMORY SPACE \$15 0000

The SP uses 4 Kbytes of external 8-bit-wide dual-ported RAM (DPR) to hold a frame status table. One port of this DPR is directly accessible and appears as 4 Kbytes of 8-bit memory spread out into 16 Kbytes of 32-bit-wide address space one memory byte per 32-bit access word beginning at address \$15 0000. Reading this memory during a DMA operation should only be performed by the Status Collector for a Status Collection Operation; host access will severely degrade performance and is strongly discouraged. Writing this memory while the SP is running will have unpredictable results and must not be allowed to occur.

[SP Frame Status Memory Description to be copied here from SP Document by Technical Publications]

3.3.14 SP PACKET STATUS MEMORY SPACE \$16 0000

The SP uses an external 16k x 32-bit (64 Kbytes) DPR to hold a frame status table. One port of this DPR is directly accessible and appears beginning at address \$16 0000. Reading this memory during a DMA operation should only be performed by the Status Collector for a Status Collection Operation; host access will severely degrade performance and is strongly discouraged. Writing this memory while the SP is running will have unpredictable results and must not be allowed to occur.

[SP Packet Status Memory Description to be copied here from SP Document by Technical Publications]

3.3.15 SP INTERNALLY-ACCESSIBLE MEMORY SPACE \$18 0000

The SP contains various internal and internally-accessible external 8- and 16-bit-wide memories. These appear spread out to various extents into 256 Kbytes of 32-bit-wide address space one memory byte or 16-bit-word per 32-bit access word beginning at address \$18 0000. Some of these memories can be dumped into the SP Data FIFOs. Those FIFOs have programmable almost-full and almost-empty flags which are used by the Baseboard Interface and SP, and their corresponding offset registers can be indirectly written through this space. These memories are not accessible via Status Collection; host access of these memories during a DMA operation will severely degrade performance and is strongly discouraged. Writing them while the SP is running will have unpredictable results and must not be allowed to occur.

[SP Internally Accessible Memory Descriptions to be copied here from SP Document by Technical Publications]

SECTION 4 HARDWARE INSTALLATION

4.1 INTRODUCTION

The RLP has a total of seven I/O interfaces: four on the I/O panel which is generally accessible from the rear of the PCI chassis, and three not on the I/O panel which are only accessible from inside the chassis.

This section provides an overview of the RLP hardware and installation.

4.2 HARDWARE ELEMENTS

4.2.1 I/O PANEL

The four I/O panel interfaces operate through one LED and five input connectors as depicted in Figure 4-1.

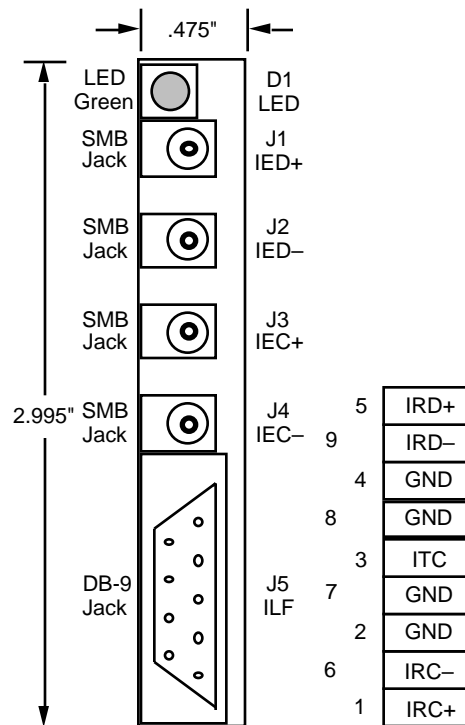
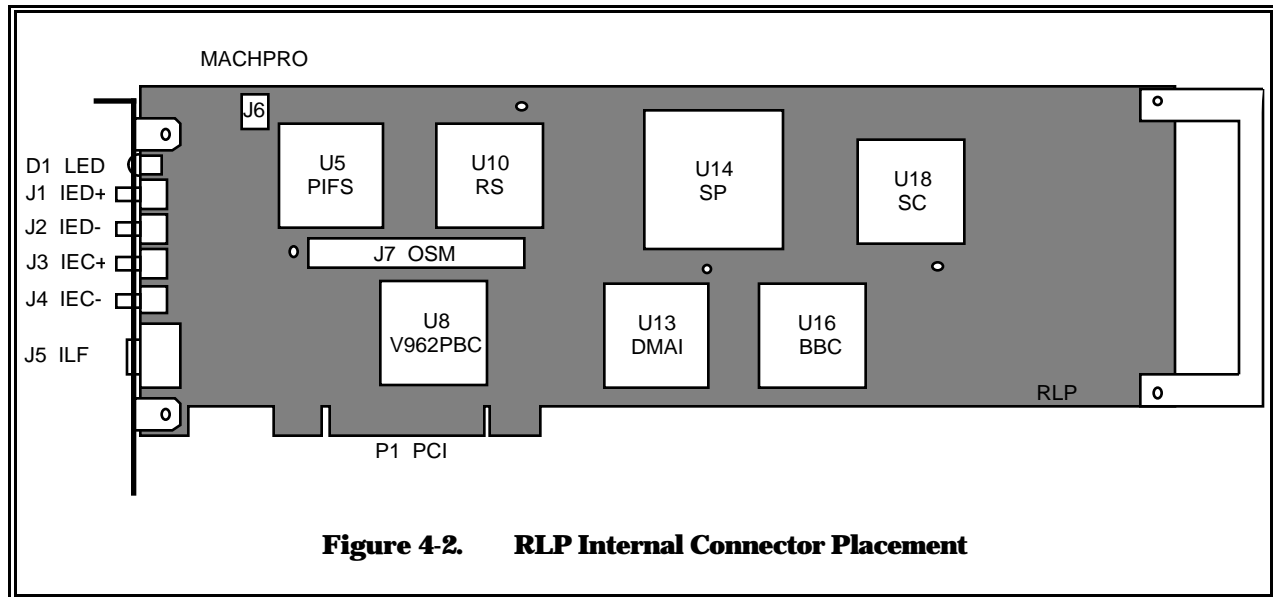


Figure 4-1. RLP I/O Panel

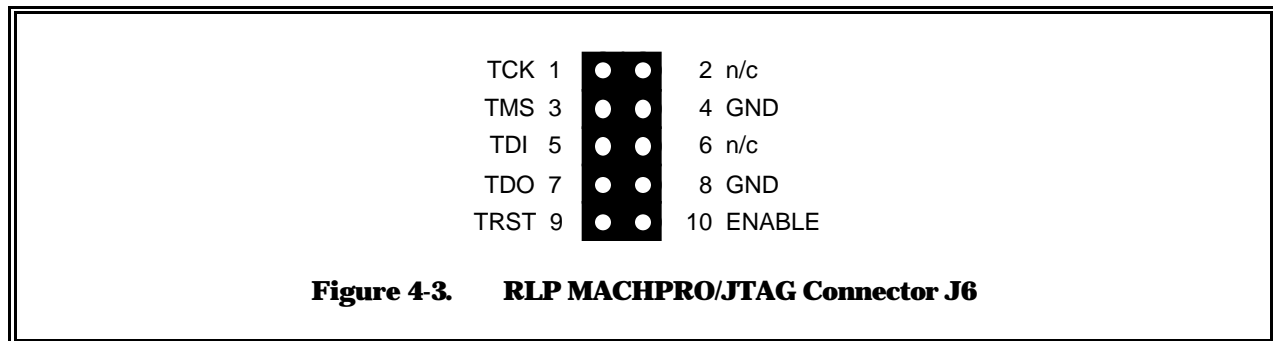
4.2.2 INTERNAL CONNECTORS

The RLP contains three internal connectors: the MACHPRO connector, the Optional Sorting Module connector, and the PCI connector. The placement of the internal connectors appears in Figure 4-2.



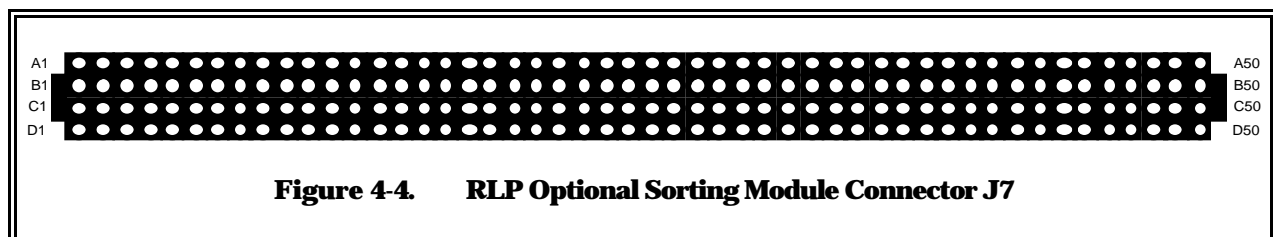
4.2.2.1 MACHPRO/JTAG Connector J6

The MACHPRO/JTAG connector is a 10-pin socket strip (2 rows of 5 sockets each) as depicted in Figure 4-3.



4.2.2.2 Optional Sorting Module Connector J7

The Optional Sorting Module connector is a 200-pin socket strip (4 rows of 50 sockets each) as depicted in Figure 4-4.



4.2.2.3 PCI Connector P1

The PCI connector is a 124-pin double-sided board-edge plug as depicted in Figure 4-5.

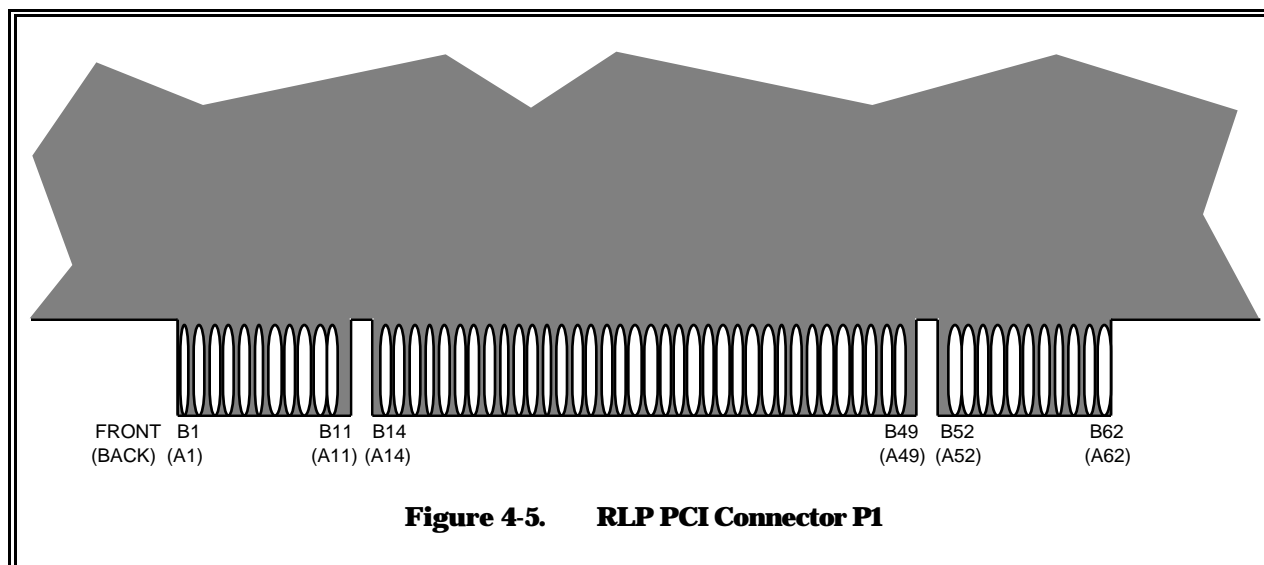


Figure 4-5. RLP PCI Connector P1

4.2.3 JUMPERS AND SWITCHES

None.

4.3 INTERFACES

The RLP has a total of 7 interfaces: LED, ECL data input, RS-422/485 data input, external timecode reference clock input, MACHPRO/JTAG, Optional Sorting Module, and PCI.

4.3.1 LED

The I/O panel contains one green LED (D1); see Figures 4-1 and 4-2 and Table 4-1. Its operation is entirely under software control – writing a 0 to bit 1 of the Baseboard Interface Control register \$04 0000 turns it on, and writing a 1 turns it off. It can be useful for testing, condition indication, "heartbeat" indication, or simple power. Upon reset it defaults to "on".

Table 4-1. RLP I/O Panel Interfaces

Ref	Name	Description		
D1	LED	LED, controlled by the host (software), green		
J1	IED+	SMB jack, ECL positive or single-ended data input, 50 impedance		
J2	IED-	SMB jack, ECL negative data input, 50 impedance		
J3	IEC+	SMB jack, ECL positive or single-ended clock input, 50 impedance		
J4	IEC-	SMB jack, ECL negative clock input, 50 impedance		
J5	ILF	DB-9 jack, low-frequency (10 MHz) input signals:		
		Pin	Signal	Description
		1	IRC+	RS-422/485 positive clock input, 120 impedance
		6	IRC-	RS-422/485 negative clock input, 120 impedance
		5	IRD+	RS-422/485 positive data input, 120 impedance
		9	IRD-	RS-422/485 negative data input, 120 impedance
		3	ITC	TTL 10 MHz timecode reference clock signal, 50 impedance
		2,4,7,8	GND	signal ground

4.3.2 ECL DATA INPUT INTERFACE

The I/O panel contains four industry-standard SMB jacks (J1-4) for accepting differential or single-ended ECL signals carrying satellite telemetry data at rates of up to 300 Mbps nominal, 400 Mbps theoretical maximum; see Figures 4-1 and 4-2 and Table 4-1. The inputs are: positive data (IED+), negative data (IED-), positive clock (IEC+), and negative clock (IEC-). They are terminated to 50 Ω into -2 Volts, so they must never be grounded, and 50 Ω shielded coaxial cable should be used. For differential signals, all four inputs must be used and bit 2 of the Baseboard Interface Control register \$04 0000 must be set to 1. For single-ended clock (regardless of data), the clock signal must be connected to IEC+ while IEC- must be tied through a 100 Ω 1% resistor to ground (the value of bit 2 of the Baseboard Interface Control register \$04 0000 is irrelevant). For single-ended data (regardless of clock), the data signal must be connected to IED+, and then there are two options: (1) set bit 2 of the Baseboard Interface Control register \$04 0000 to 0 (in which case it is irrelevant whether or not anything is connected to IED-), or (2) tie IED- through a 100 Ω 1% resistor to ground (in which case the value of bit 2 of the Baseboard Interface Control register \$04 0000 is irrelevant). Upon reset the bit defaults to 1, or differential.

ECL data enters the PIFS through its bit-parallel-byte-serial data input port, so it must be set up appropriately. If ECL input is not being used, all inputs should be left unconnected; while connecting them will not affect performance, it may unnecessarily increase power consumption and heat dissipation.

4.3.3 RS-422/485 DATA INPUT INTERFACE

The I/O panel contains one DB-9 jack (J5) for low-frequency input signals (ILF); see Figures 4-1 and 4-2 and Table 4-1. It contains pins which will accept both RS-422 and RS-485 type satellite telemetry data signals at up to 10 Mbps nominal, sometimes higher. The inputs are: positive data (pin 5, IRD+), negative data (pin 9, IRD-), positive clock (pin 1, IRC+), and negative clock (pin 6, IRC-). They are terminated with 120 Ω to their differential counterparts, so 120 Ω shielded twisted-pair cable should be used. Low-rate data enters the PIFS through its bit-serial data input port number 0, so it must be set up appropriately. Low-rate inputs left connected when unused have no effect.

4.3.4 EXTERNAL TIMECODE REFERENCE CLOCK INPUT INTERFACE

The PIFS generates a timecode from a software-loaded register and a 10 MHz clock. An external high-stability, high-accuracy TTL-level 10 MHz timecode reference clock signal may be supplied through the I/O panel DB-9 jack J5 for low-frequency input signals (ILF) pin 3 (ITC); see Figures 4-1 and 4-2 and Table 4-1. This input is terminated with 50 Ω to ground, so 50 Ω shielded coaxial cable should be used. The signal will be supplied to the PIFS in lieu of the onboard oscillator if bit 5 of the Baseboard Interface Control register \$04 0000 is set to 1. Upon reset the bit defaults to 0, or internal oscillator. A signal left connected when unused has no effect.

4.3.5 MACHPRO/JTAG INTERFACE

The Baseboard Interface, DMA Interface, and Status Collector are implemented in AMD MACH CPLDs which can be programmed, reprogrammed, and tested in-circuit. This is done through the MACHPRO/JTAG connector J6; see Figures 4-2 and 4-3 and Table 4-2. Further information is available from the reference documents listed in Section 1.4 and the RLP schematics.

Table 4-2. RLP MACHPRO/JTAG Connector J6 Signals

Pin	Signal	Description
1	TCK	JTAG Test / AMD MACH Programming Clock
3	TMS	JTAG Test / AMD MACH Programming Mode Select
5	TDI	JTAG Test / AMD MACH Programming Data In
7	TDO	JTAG Test / AMD MACH Programming Data Out
9	TRST	JTAG Test / AMD MACH Programming Reset
10	ENABLE	AMD MACH Programming Enable
4,8	GND	signal ground
2,6	n/c	not connected

4.3.6 OPTIONAL SORTING MODULE INTERFACE

Functionality can be added to the RLP using an optional mezzanine board. All signals deemed useful for this purpose are present in the Optional Sorting Module connector J7; see Figures 4-2 and 4-4 and Table 4-3. The connector also provides convenient access to a large number of signals for testing purposes. Refer to the RLP schematics for further information.

Table 4-3. RLP Optional Sorting Module Connector J7 Signals

Signal	Pin	Signal	Pin	Signal	Pin	Signal	Pin
GND	A1	LA24	B1	PK1AE	C1	STOE	D1
FEOD	A2	LA23	B2	PK2AE	C2	FASTCKI	D2
FD7	A3	VDD	B3	PK3AE	C3	LADS	D3
FD6	A4	LA22	B4	PRJAE	C4	LREADY	D4
FD5	A5	LA21	B5	VCC	C5	LW_R	D5
FD4	A6	LA20	B6	STAE	C6	LHOLD	D6
FD3	A7	LA19	B7	ST1E	C7	GND	D7
FD2	A8	LA18	B8	ST2E	C8	LHOLDA	D8
VCC	A9	LA17	B9	ST3E	C9	LCLK	D9
FD1	A10	LA16	B10	FR1E	C10	LBLAST	D10
FD0	A11	GND	B11	FR2E	C11	LBTERM	D11
LD31	A12	LA15	B12	MISCE	C12	TDI	D12
LD30	A13	LA14	B13	VDD	C13	TDO	D13
LD29	A14	LA13	B14	BITE	C14	TCK	D14
LD28	A15	LA12	B15	FRJE	C15	VCC	D15
LD27	A16	LA11	B16	PK1E	C16	TMS	D16
VDD	A17	LA10	B17	PK2E	C17	TRST	D17
LD26	A18	LA9	B18	PK3E	C18	ENABLE	D18
LD25	A19	VCC	B19	PRJE	C19	FRCKO	D19
LD24	A20	LA8	B20	STE	C20	SCL	D20
LD23	A21	LA7	B21	GND	C21	SDA	D21
LD22	A22	LA6	B22	FLD	C22	INT	D22
LD21	A23	LA5	B23	ST1RE	C23	VDD	D23
LD20	A24	LA4	B24	ST2RE	C24	OSMPRES	D24
GND	A25	LA3	B25	ST3RE	C25	RESET	D25
LD19	A26	LA2	B26	FR1RE	C26	FRCKI	D26
LD18	A27	VDD	B27	FR2RE	C27	NC	D27

LD17	A28	ST1AF	B28	MISCRE	C28	NC	D28
LD16	A29	ST2AF	B29	VCC	C29	GND	D29
LD15	A30	ST3AF	B30	BITRE	C30	NC	D30
LD14	A31	FR1AF	B31	FRJRE	C31	GND	D31
LD13	A32	FR2AF	B32	PK1RE	C32	NC	D32
VCC	A33	MISCAF	B33	PK2RE	C33	GND	D33
LD12	A34	BITAF	B34	PK3RE	C34	NC	D34
LD11	A35	GND	B35	PRJRE	C35	GND	D35
LD10	A36	FRJAF	B36	STRE	C36	NC	D36
LD9	A37	PK1AF	B37	VDD	C37	GND	D37
LD8	A38	PK2AF	B38	ST1OE	C38	NC	D38
LD7	A39	PK3AF	B39	ST2OE	C39	VCC	D39
LD6	A40	PRJAF	B40	ST3OE	C40	NC	D40
VDD	A41	STAF	B41	FR1OE	C41	GND	D41
LD5	A42	ST1AE	B42	FR2OE	C42	NC	D42
LD4	A43	VCC	B43	MISCOE	C43	GND	D43
LD3	A44	ST2AE	B44	BITOE	C44	NC	D44
LD2	A45	ST3AE	B45	GND	C45	GND	D45
LD1	A46	FR1AE	B46	FRJOE	C46	NC	D46
LD0	A47	FR2AE	B47	PK1OE	C47	VDD	D47
LA27	A48	MISCAE	B48	PK2OE	C48	NC	D48
LA26	A49	BITAE	B49	PK3OE	C49	NC	D49
LA25	A50	FRJAE	B50	PRJOE	C50	NC	D50

4.3.7 PCI BUS INTERFACE

The RLP interfaces directly with a host computer through the industry-standard PCI bus connector P1; see Figures 4-2 and 4-5 and Table 4-4. Further information is available from the reference documents listed in Section 1.4 and the RLP schematics.

Table 4-4. RLP PCI Bus Connector P1 Signals

Signal	Pin	Signal	Pin	Signal	Pin	Signal	Pin
TRST	A1	AD16	A32	VN12_0	B1	AD17	B32
VP12_0	A2	VDD	A33	TCK	B2	CBE2	B33
TMS	A3	FRAME	A34	GND	B3	GND	B34
TDI	A4	GND	A35	TDO	B4	IRDY	B35
VP5_0	A5	TRDY	A36	VP5_0	B5	VDD	B36
INTA	A6	GND	A37	VP5_0	B6	DEVSEL	B37
INTC	A7	STOP	A38	INTB	B7	GND	B38
VP5_0	A8	VDD	A39	INTD	B8	LOCK	B39
NC	A9	SDONE	A40	PRSNT1	B9	PERR	B40
NC	A10	SBO	A41	NC	B10	VDD	B41
NC	A11	GND	A42	PRSNT2	B11	SERR	B42
KEYWAY	A12	PAR	A43	KEYWAY	B12	VDD	B43
KEYWAY	A13	AD15	A44	KEYWAY	B13	CBE1	B44
NC	A14	VDD	A45	NC	B14	AD14	B45
RST	A15	AD13	A46	GND	B15	GND	B46
NC	A16	AD11	A47	CLK	B16	AD12	B47
GNT	A17	GND	A48	GND	B17	AD10	B48

GND	A18	AD9	A49	REQ	B18	M66EN	B49
NC	A19	KEYWAY	A50	NC	B19	KEYWAY	B50
AD30	A20	KEYWAY	A51	AD31	B20	KEYWAY	B51
VDD	A21	CBE0	A52	AD29	B21	AD8	B52
AD28	A22	VDD	A53	GND	B22	AD7	B53
AD26	A23	AD6	A54	AD27	B23	VDD	B54
GND	A24	AD4	A55	AD25	B24	AD5	B55
AD24	A25	GND	A56	VDD	B25	AD3	B56
IDSEL	A26	AD2	A57	CBE3	B26	GND	B57
VDD	A27	AD0	A58	AD23	B27	AD1	B58
AD22	A28	NC	A59	GND	B28	NC	B59
AD20	A29	REQ64	A60	AD21	B29	ACK64	B60
GND	A30	VP5_0	A61	AD19	B30	VP5_0	B61
AD18	A31	VP5_0	A62	VDD	B31	VP5_0	B62

4.4 INSTALLATION GUIDELINES

Follow the standard procedures for installing a PCI expansion card in a PCI host chassis. In brief, this involves having the power off but not unplugged (maintaining the integrity of the ground), properly grounding the installer and surroundings, opening the chassis, removing an I/O panel filler plate if necessary, inserting the RLP, fastening the I/O panel screw, closing the chassis, and lastly connecting all appropriate input cables. The driver and application software must also be installed.

4.5 ENVIRONMENTAL REQUIREMENTS

The RLP is a standard commercial-grade computer product. It will operate from 0 to +70 degrees Centigrade. It is not designed to withstand any unusual levels of physical shock or nuclear or electromagnetic radiation. It is ESD-sensitive and all appropriate precautions must be taken before and during its handling. It is only specified to operate in a PCI Revision 2.1-compliant host platform. Any attempt to operate the RLP in a manner inconsistent with these specifications will void any warranty, express or implied, and will run the risk of damage to the RLP.

SECTION 5 OPERATING PRINCIPLES

5.1 INTRODUCTION

During normal operation, the RLP is basically seen by the satellite telemetry bitstream provider as a data sink, and by the PCI host computer as a PCI data source. Since the RLP has no MPU of its own, it behaves as a simple data flow device with straightforward operation: set the card up prior to a data session, and service it (move data off the card as it becomes available) during the session. The most critical aspect is in servicing the card fast enough and efficiently enough to avoid data loss.

This section outlines procedures to set up, control, and monitor the RLP and the flow of data.

5.2 PRELIMINARIES

For the RLP to function, the serial electrically-eraseable programmable ROM (EEPROM) and the three CPLDs must be programmed. In particular, the EEPROM must be programmed out-of-circuit prior to board assembly. While it is reprogrammable in-circuit via the PCI bus, the board must first function on the bus to make reprogramming possible, but because PCI configuration is read from the EEPROM by the PBC, this cannot happen without valid content. The CPLDs can be programmed out-of-circuit prior to assembly or in-circuit afterwards via the MACHPRO connector. In all cases, device reprogramming is a development function only and should not be encountered during normal end-user operation; further information is available from the documents listed in Section 1.4 .

5.3 SETUP

Upon power-up or reset of the PCI host computer, the RLP's PBC automatically reads PCI configuration data from its onboard ROM and sets up its PCI interface. Then depending on the host, either the operating system (OS) or the driver software reads the RLP's PCI configuration and initializes itself accordingly. At this point the RLP is ready to be set up for a data session by the application software.

5.3.1 RESETS AND INPUTS

After power-up or reset of the PCI host computer, most of the RLP circuits are held in reset by the Baseboard Interface. The only circuits that are accessible in this condition are the DMA-chaining memory, the Baseboard Interface, and the PBC. All the resets can be released at this point through the Baseboard Interface Control register unless activity exists at the ECL inputs, in which case the ECL Serial-to-Parallel Converter Reset bit should be left asserted until the data session is ready to begin. Activity on the RS-422/485 and optional timecode reference clock inputs has no effect. Refer to Section 3.3.2.1 and the schematics for more details.

5.3.2 PCI INTERFACE

The PBC must be set up. Local bus byte enables are not used (always enabled) and local bus parity checking is not used. The PBC runs off a 33 MHz local bus clock. It inputs the interrupt from the Baseboard Interface on INTC, inputs the interrupt from the Optional Sorting Module on INTD, outputs the interrupt to the PCI bus on INTA, and does not use INTB (which is connected to the PCI bus). The PCI JTAG Boundary Scan bus is connected to the PIFS and SP only. Refer to the documents listed in Section 1.4 and the schematics for more details.

5.3.3 INPUT INTERFACE

If ECL input is being used, single-ended or differential mode must be selected with the ECL Input Type Select bit of the Baseboard Interface Control register; default is differential. If an external timecode reference 10 MHz clock is being used, PIFS Timecode Reference Clock Select bit of the same register must be set; default is internal oscillator. Refer to Section 3.3.2.1 and the schematics for more details.

5.3.4 PIFS

The PIFS must be set up according to user and data session requirements; it has 32 contiguous 32-bit registers. If the input source is ECL, use the PIFS parallel data input port. If the input source is RS-422/485, use the PIFS serial input port number 0. Data can also be injected by the host through a PIFS register. Data can be output from the PIFS either 8 or 16 bits at a time; 16 is normally suggested for optimal performance when outputting to the RS, but 8 bits must be used if the data is routed around the RS and SP using FS-Direct mode discussed in Section 5.3.5, since that is the size of the destination Data FIFO. The timecode reference clock is 10 MHz, and the processing clock is 50 MHz. Refer to the documents listed in Section 1.4 and the schematics for details.

5.3.5 DATA ROUTING MODE

Data may be routed through the RLP in three ways: it can go from the PIFS through the RS then through the SP into the Data FIFOs (SP Mode), from the PIFS through the RS into a Data FIFO (RS-Direct Mode), or from the PIFS into a Data FIFO (FS-Direct Mode). This must be selected with the Processing Mode bits of the Baseboard Interface Control register; default is SP Mode. If the RS and/or SP are bypassed, it is suggested that they be set up so they do not inadvertently read from their input FIFOs or write to their output FIFOs. Refer to Section 3.3.2.1 and the schematics for more details.

5.3.6 RS

The RS must be set up according to user and data session requirements; it has forty 16-bit registers appearing in 40 contiguous 32-bit locations, one register per location. The input data size of 8 or 16 bits must match the output data size set up in the PIFS. Data output from the RS can be either 8 or 16 bits wide; 16 is normally suggested for optimal performance when outputting to the SP, but if the data is routed around the SP using RS-Direct mode discussed in Section 5.3.5, it must be 8 bits since that is the size of the destination Data FIFO. The RS has no external memory and does not perform routing. It runs off the 50 MHz processing clock. Refer to the documents listed in Section 1.4 and the schematics for more details.

5.3.7 SP

The SP must be set up according to user and data session requirements; it has one-hundred-and-twelve 16-bit registers appearing in 112 contiguous 32-bit locations, one register per location. Input data size of 8 or 16 bits must match the output data size set up in the RS. Some of the SP's internal and external memories must be set up: it has 512k x 32 bits of 20 ns Packet Data Buffer SRAM, 512k x 32 bits of 20 ns Packet Lookup Table SRAM, 512k x 8 bits of 20 ns Frame Lookup Table SRAM, 16k x 32 bits of Packet Status Table DPR, and 4k x 8 bits of Frame Status Table DPR.

The SP outputs to four 32k x 8 bit Packet Service Data FIFOs and eight 32k x 8 bit Frame Service Data FIFOs with programmable flags that must be programmed through the SP (discussed in Section 5.3.11).

It runs off the 50 MHz processing clock. Refer to the documents listed in Section 1.4 and the schematics for details.

5.3.8 DMA INTERFACE

The only setup the DMA Interface has is two bits in its Control register. The DMA Interface Transfer FIFO Partial Reset bit resets all but the programmable flags when set to 0; after power-up or reset of the PCI host computer, it defaults to not-reset. The Data Byte Order Mode bit selects whether the chronologically first telemetry data byte appears in PCI data bits 31-24 with subsequent bytes following in descending byte positions, when set to 0, or bits 7-0 with subsequent bytes following in ascending byte positions, when set to 1; after power-up or reset of the PCI host computer, it defaults to bits 31-24. Refer to Section 3.3.4.5 for more information.

The DMA Interface buffers data in four 32k x 8-bit FIFOs which appear to the user as a single 32k x 32-bit FIFO. This has programmable flags which must be programmed through the DMA Interface (discussed in Section 5.3.11).

5.3.9 STATUS COLLECTOR

The Status Collector has no setup, but its 32k x 8-bit output Data FIFO has programmable flags that must be programmed through the Status Collector Data FIFO Input Space (discussed in Section 5.3.11).

5.3.10 INTERRUPTS

The same conditions, signals, and registers used to set up, control, and monitor interrupts are also used for polling, the only difference being that for polling, the assertion of the baseboard interrupt signal is disabled. Therefore where polling is not explicitly mentioned in the discussion, it is implied.

The Baseboard Interface Control register Interrupt Source Indication Mode bit must be set up. If this bit is set to 1 or “masked”, the signals indicated in the Interrupt Source registers appear masked as specified in the Interrupt Source Mask registers, and thus show precisely the source(s) that are generating the baseboard interrupt. If set to 0 or “raw”, all the interrupt source signals appear in the actual state in which they exist on the board regardless of whether or not they are masked, and thus regardless of whether or not they are being used to generate the baseboard interrupt. Upon power-up or reset of the PCI host computer, this bit defaults to raw. See Section 3.3.2.1 for more details.

The 54 interrupt source masks must be set up. These reside in Baseboard Interface Interrupt Source Mask registers. If a source is masked, it will not be used to generate the baseboard interrupt, and if the Interrupt Source Indication Mode is also set to masked, the interrupt source will always appear as deasserted in its Interrupt Source register regardless of whether or not it is really asserted. Upon power-up or reset of the PCI host computer, all interrupt sources default to masked. See Section 3.3.2 for more details.

5.3.11 PROGRAMMABLE FIFO FLAGS

All 13 Data FIFOs and the Transfer FIFO have programmable almost-full and almost-empty flags which can be reprogrammed differently from their defaults. Refer to the documents listed in Section 1.4 for the basic default and programming information. To program any of these FIFOs, the Universal FIFO Programmable Flag Load bit of the Baseboard Interface Control register must be set to 0 or “load” while programming data is being written. At all other times it must be set to 1 or

“normal”. After power-up or reset of the PCI host computer, this bit defaults to “load”. See Section3.3.2.1 for more details.

The twelve Data FIFOs on the outputs of the SP are programmed through the SP. This involves writing the programming data into SP memory and then causing the SP to dump that memory into the selected FIFOs (with the Baseboard Interface Control register Universal FIFO Programmable Flag Load bit set). The SP can individually program each FIFO to different values, or simultaneously program multiple FIFOs to the same values. Refer to the documents listed in Section1.4 for more information. The SP uses the almost-full flags of these FIFOs to determine when to stop writing telemetry data into them, so they must never be programmed to higher than nine bytes below full; the defaults are sufficient for these flags. If they are programmed lower, all but nine bytes of the space above that will go unused, effectively creating smaller FIFOs. For service interrupts, the half-full and programmable-almost-empty flags are the most useful. Half-full is a good place to start, and if performance or priority issues warrant, almost-empty can be programmed appropriately and used instead or in addition. For almost-empty and empty flags, it is the inverse that is detected to cause an interrupt, or not-almost-empty and not-empty. In this way the almost-empty flag acts as another almost-full flag. See Section3.3.2 for more details.

The Data FIFO on the output of the Status Collector is programmed directly through the Status Collector Output Data FIFO Input Space (with the Baseboard Interface Control register Universal FIFO Programmable Flag Load bit set). On each 32-bit write to the space, the least-significant 9 bits are written into the FIFO. The Status Collector uses the almost-full flag to determine when to stop writing status data, so it must never be programmed to higher than four bytes below full; the default is sufficient for this flag. If it is programmed lower, all but four bytes of the space above that will go unused, effectively creating a smaller FIFO. If packet and frame status are not being collected, the maximum possible amount of status data per collection is so small that the Status Collection Done indicator is the only useful service interrupt source; the FIFO flags can be ignored. Otherwise, the half-full and programmable-almost-empty flags are the most useful. Half-full is a good place to start, and if performance or priority issues warrant, almost-empty can be programmed appropriately and used instead or in addition. For almost-empty and empty flags, it is the inverse that is detected to cause an interrupt, or not-almost-empty and not-empty. In this way the almost-empty flag acts as another almost-full flag. See Section3.3.2 for more details.

The 32k x 32-bit DMA Interface Transfer FIFO is composed of four 32k x 8-bit FIFOs in parallel, and they all must be programmed to the same value through the DMAI Transfer FIFO Input Space (with the Baseboard Interface Control register Universal FIFO Programmable Flag Load bit set). The DMA Interface uses the almost-full flags to determine when to stop transferring telemetry data into the FIFO, so it must never be programmed to higher than four bytes below full; the default is sufficient for this flag. If it is programmed lower, all but four bytes of the space above that will go unused, effectively creating a smaller FIFO. The DMA Interface uses the almost-empty flag to perform bursting for the PCI DMA. Initially, while the amount of data in the Transfer FIFO is below the almost-empty threshold and a Byte Transfer is in progress, local bus reads from PCI space are held off. Then after the threshold has been exceeded, while the Transfer FIFO is not empty, reads are fulfilled, forming a maximally-dense PCI burst. If the Transfer FIFO becomes empty before the Transfer is complete, the DMA Interface once again holds off reads to build up another burst. Therefore the almost-empty flag sets the minimum size of the PCI burst and should be set to the largest value tolerable; 1 Kbytes to 4 Kbytes may be a good place to start. That works out to 256 B to 1 Kbytes per each of the four component FIFOs. Each 32-bit write to the space writes the least-significant 8 bits to all four FIFOs simultaneously, so programming all is like programming just one.

5.3.12 BASEBOARD INTERFACE

The only setup for the Baseboard Interface involves the reset, input, routing, and interrupt functions discussed in Section 5.3.1, Section 5.3.3, Section 5.3.5, and Section 5.3.10, respectively.

5.4 OPERATION

When the setup is complete, the PIFS inputs are enabled, the ECL Serial-to-Parallel Converter Reset is released (if used), and the baseboard interrupt is enabled (if used), the RLP is ready to begin a data session. The typical service cycle consists of four phases: waiting for an accumulation of data, initiating a data transfer, waiting for completion of the DMA, and then rechecking for more data. At the conclusion of a session, any data remaining in-process may be flushed. If interrupt latency is found to limit performance unacceptably, polling must be used when waiting for data; whether or not polling is superior to interrupts for detecting when a DMA is complete is system-dependent and must be determined empirically.. The same conditions, signals, and registers used to set up, control, and monitor interrupts are also used for polling, the only difference being that for polling, the assertion of the baseboard interrupt signal (and perhaps the DMA-done interrupt) is disabled. Therefore where polling is not explicitly mentioned in the discussion, it is implied.

5.4.1 WAIT FOR DATA

While no DMA is in progress, software waits for enough data to accumulate in one or more 8-bit-wide Data FIFOs to initiate a Byte Transfer into the 32-bit-wide Transfer FIFO. It may wait for an interrupt or may poll the Baseboard Interface Interrupt Source register(s) A and perhaps B. During this phase it may also check a heartbeat timer and toggle the LED by changing the Baseboard Interface Control register LED bit, or initiate a Status Collection by writing a request to the Status Collector Status Request register.

5.4.2 INITIATE A TRANSFER

Upon receiving an interrupt, the software first clears the DMA Done interrupt in the PBC and disables the baseboard interrupt with the Baseboard Interface Interrupt Source Mask register A Baseboard Interrupt Enable bit,, then reads the Baseboard Interface Interrupt Source register A and perhaps B to ascertain the source. Software has the subset of the 54 possible sources that it did not mask during setup prioritized by importance, for example: Foremost FIFO Overflow indicator first, followed by the RS-to-SP FIFO full flag, then the DMAI FIFO full flag, then the SP Data FIFO almost-full flags, then the SP Data FIFO almost-empty flags, then the Status Data FIFO almost-full flag, then the Status Data FIFO almost-empty flag. Depending on the condition, software may elect to take special measures such as clearing some FIFOs in order to prevent data loss in others. It might also maintain a record of previous sources that have not been serviced in order to detect when lower-priority sources have been waiting too long; it can then increase their relative priority to facilitate them getting serviced.

Once software determines which source to service, it sets up a Byte Transfer by writing a Byte Transfer Request to an unoccupied DMA Interface Control registers. The Transfer Size is equal to (or less than) the amount of data indicated to be in the Data FIFO by the source FIFO flag. If this is the first cycle of the session or the other Byte Transfer channel is unoccupied, a second Byte Transfer is also initiated if possible.

After the Byte Transfer(s) have been initiated, software sets up one DMA in the PBC. For optimum performance, the DMA is for the last previously-initiated Byte Transfer, if such exists, or otherwise for the first currently-initiated Byte Transfer. By always being one Byte Transfer

ahead of the DMAs, the DMA Interface circuit will be kept busy instead of falling idle as it would after completing the last DMA of the previous cycle and waiting for the first Byte Transfer of the current cycle. Using this form of pipelining, optimum performance can be achieved.

5.4.3 WAIT FOR DMA COMPLETION

A DMA transaction begins as soon as it is initiated. While a DMA transaction is ongoing, PCI bus traffic must be kept to an absolute minimum to ensure optimum performance. This means the host should not attempt to access the RLP at all until the DMA is complete and it receives the DMA-Done interrupt. That is, unless it is polling the PBC to ascertain this condition. It is unclear which has the larger impact, interrupt latency of the DMA-Done interrupt or bus interference of polling; this must be experimentally-determined on a platform-by-platform basis.

During this time the host is free to perform local processing, although it will most likely be called upon to transfer data already in host memory to an I/O device for storage.

5.4.4 RECHECK FOR DATA

When the DMA has completed, the software clears the DMA Done interrupt in the PBC, and if appropriate, clears the foremost FIFO overflow condition with the Baseboard Interface Control register Clear Foremost FIFO Overflow bit. It then checks the Baseboard Interface Interrupt Source register(s) so that no time is wasted if data exists to transfer. In this case, it goes back to the “Initiating a Transfer” phase. Otherwise, it re-enables the baseboard interrupt with the Baseboard Interface Interrupt Source Mask register A Baseboard Interrupt Enable bit and then goes back to the “Waiting for Data” phase.

5.4.5 FLUSH REMAINING DATA

Eventually after a data session has terminated, all of the interrupt sources have been serviced and none remain. At this point some data may still exist in the Data FIFOs. It is easily extracted by performing one additional Byte Transfer per Data FIFO with a Transfer Size equal to that indicated by the lowest interrupt source that remains unasserted. After each Byte Transfer, the corresponding DMA Interface Fill Count register is read because it indicates the amount of filler bytes (copies of the last valid data byte) that was necessary to append to the end of the data to fulfill the requested Byte Transfer. A simple subtraction yields the exact amount of data that has been flushed from the Data FIFO.

If the SP was used and was set up to operate on partial packet pieces, some of those may remain in SP memories. If these are desired, they must be extracted individually directly from the SP. More information is available from the documents referenced in Section 1.4.

SECTION 6

SCHEMATIC DESCRIPTION

6.1 INTRODUCTION

In addition to the customary two assembly and one drill drawings, the RLP schematic has 13 pages. All drawings are provided in Appendix A. There are also three CPLDs and one ROM, and their programs are listed in Appendix B. This section describes the drawings and program listings.

6.2 DRAWING DIRECTORY

The RLP drawings are listed in Table 6-1.

Table 6-1. RLP Drawing Directory

Sheet	Title
-	Top Assembly
-	Bottom Assembly
-	Drill Master
1	Table of Contents
2	Input Connectors & Interface, LED, PIFS ASIC & FIFO, Processing & Timestamp Clocks
3	RS ASIC & FIFO, SP Packet Buffer Memory
4	SP ASIC, Frame Lookup & Status Memories
5	SP Packet Lookup & Status Memories
6	SP Frame Service FIFOs
7	FS-/RS-Direct MUX, SP Frame & Packet Service FIFOs, Status FIFO
8	DMA Interface CPLD, FIFOs, Clock
9	Baseboard Interface CPLD, DMA-Chaining Memory, Local Bus Isolation Switch, Temperature Sensor, A/D Converter, Status Collector CPLD
10	Optional Sorting Module Connector, PCI Bus Interface IC & Connector, Local Bus Clock, MACHPRO Connector
11	+/-3.3V, -5.2V, +/-12V Power, Terminations, Bypasses
12	+5V Power, Current Sensor, Bypasses
13	Test Points

6.3 DRAWING DESCRIPTION

6.3.1 TOP ASSEMBLY

Used by assembly to place the devices on the front side of the RLP printed circuit board

6.3.2 BOTTOM ASSEMBLY

Used by assembly to place the devices on the back side of the RLP printed circuit board

6.3.3 DRILL MASTER

Used by fabrication to cut and drill the RLP printed circuit board

6.3.4 SHEET 1

RLP schematic table of contents, list of design revisions

6.3.5 SHEET 2

SMB (ECL) and DB-9 (RS-422/485) input connectors, LED, ECL input circuit, RS-422/485 input circuit, PIFS ASIC, PIFS output FIFO, internal 10 MHz timecode reference clock oscillator, 50 MHz processing clock oscillator and distribution circuit

6.3.6 SHEET 3

RS ASIC, RS output FIFO, SP packet buffer SRAMs

6.3.7 SHEET 4

SP ASIC, SP frame lookup SRAM, SP frame status DPR

6.3.8 SHEET 5

SP packet lookup SRAMs, SP packet status DPRs

6.3.9 SHEET 6

SP output frame service FIFOs

6.3.10 SHEET 7

SP/RS-Direct/FS-Direct multiplexer data routing circuit, multiplexed output data FIFO, SP output packet service FIFOs, Status Collector output data FIFO

6.3.11 SHEET 8

DMA Interface CPLD, DMAI transfer FIFOs, 60 MHz DMAI byte transfer clock oscillator and distribution circuit

6.3.12 SHEET 9

Baseboard Interface CPLD, DMA-chaining SRAMs, local telemetry/status bus isolation switch, Status Collector CPLD, temperature sensor, A/D converter

6.3.13 SHEET 10

Optional Sorting Module connector, V962PBC ASIC, PCI bus connector, PCI configuration EEPROM, 33 MHz local bus clock oscillator & distribution circuit, AMD MACHPRO/JTAG connector, local bus pull resistors

6.3.14 SHEET 11

Clock line terminations, +3.3V bypass capacitors, +/-12V bypass capacitors, -5.2V power converter & bypass capacitors, -3.3V power converter & bypass capacitors, ECL terminations

6.3.15 SHEET 12

+5V bypass capacitors, current sensor circuit

6.3.16 SHEET 13

Test points for signals not available on Optional Sorting Module connector

6.4 PROGRAM LISTING DIRECTORY

The RLP programmable device program listings are listed in Table 6-2.

Table 6-2. RLP Program Listing Directory

Pages	Description
5	Baseboard Interface Program
6	DMA Interface Program
8	Status Collector Program
1	PCI Configuration ROM Data

6.5 PROGRAM LISTING DESCRIPTION**6.4.1 BASEBOARD INTERFACE**

The Baseboard Interface is an AMD MACH465/466 CLPD; see the references listed in Section 1.4 for device information.

It inputs 54 signals that could cause a PCI interrupt, indicates them in a pair of registers, allows them to be individually masked in another pair of registers, and generates the interrupt signal for the PBC to use to generate the PCI Interrupt.

The Baseboard Interface provides control of all device resets, FIFO programming signals, LED state, differential/single-ended ECL data input mode, internal/external PIFS timecode reference clock select, and SP/RS-Direct/FS-Direct data routing mode in a single register.

The Baseboard Interface arbitrates with the Status Collector for host access to the status bus segment, and it operates the status/telemetry bus segment isolation switch.

The Baseboard Interface decodes the addresses to its register space, the overall status bus space, and the DMA-chaining RAM.

6.4.2 DMA INTERFACE

The DMA Interface is an AMD MACH465/466 CLPD; see the references listed in Section 1.4 for device information.

Upon receipt of a Data Transfer Request in one of two Control registers, the DMA Interface CPLD transfers the requested number of bytes from the requested one of thirteen 8-bit-wide source Data FIFOs and packs them into the 32-bit-wide DMAI Transfer FIFO at one byte per 60 MHz clock cycle. If the source FIFO goes empty, the DMA Interface fills the remainder of the transfer with filler bytes (copies of the last valid data byte) and counts the number of filler bytes in the corresponding one of two Fill Count registers.

Upon a local bus read of the DMAI Transfer FIFO output space, the DMA Interface controls the transaction. It supports bursting at up to the maximum local bus speed of one 32-bit word every 33 MHz local bus clock cycle. It holds off the first read until the DMAI Transfer FIFO programmable almost-empty flag goes not-almost-empty, and then fulfills reads until either the DMA is complete or the DMAI Transfer FIFO goes empty. If the DMAI Transfer FIFO goes empty before the Byte Transfer Operation that is feeding it is complete, the DMA Interface returns to the hold state (waiting for not-almost-empty) to build up for another burst.

Upon a local bus read from DMAI Data FIFO output space or a local bus write into the DMAI Transfer FIFO input space, the DMA Interface controls the transaction. It does not necessarily support this at the maximum local bus rate.

If an Optional Sorting Module is present, the DMA Interface disables itself, allowing the Module to read from the Data FIFOs and respond to local bus reads and writes from the PBC.

The DMA Interface decodes the addresses to its register space, the SP Data FIFO output space, the DMAI Transfer FIFO input space, and the DMAI Transfer FIFO output space.

6.4.3 STATUS COLLECTOR

The Status Collector is an AMD MACH465/466 CLPD; see the references listed in Section 1.4 for device information.

Upon receipt of a Status Request in the Status Request register, the Status Collector CPLD collects all the requested status from the requested devices on the status bus segment and deposits it into an 8-bit data FIFO with a local bus clock rate of 33 MHz. It does not deposit the "spacer" bytes used to align 8- and 16-bit data to 32-bit boundaries.

The Status Collector constantly and autonomously reads the temperature and current values from the ADC and stores them in a register.

The Status Collector arbitrates with the Baseboard Interface for host access of the status bus segment.

The Status Collector decodes all the addresses on the status bus segment including its register space and the Status Data FIFO input space.

6.4.4 PCI CONFIGURATION ROM

The PCI Configuration ROM is an ATMEL AT24C02 serial EEPROM; see Section 1.4 for device information.

It contains the first 128 bytes of PCI configuration space that the PBC reads to set its PCI Configuration values after power-up or reset. See the references listed in Section 1.4 for details.

ABBREVIATIONS AND ACRONYMS

This is a list of the meanings of the abbreviations and acronyms found in this document; see Table AB-1.

Table AB-1. Abbreviations and Acronyms

Term	Definition
ADC	analog-to-digital converter
AMD	Advanced Micro Devices
AOS	Advanced Orbiting Systems
ASIC	application-specific integrated circuit
BTD	bit transition density
CCSDS	Consultative Committee for Space Data Systems
CMOS	complimentary metal-oxide-semiconductor
COTS	commercial off-the-shelf
CPLD	complex PLD
CRC	cyclic redundancy code
DEC	Digital Equipment Corporation
DMA	direct memory access
DMAI	DMA Interface
DPR	dual-ported SRAM
DSDP	Desktop Satellite Data Processor
DSTD	Data Systems Technology Division
ECL	emitter-coupled logic
EEPROM	electrically-eraseable PROM
EPROM	eraseable PROM
FIFO	first-in, first-out
FS	Frame Synchronizer
GaAs	Gallium-Arsenide
GSFC	Goddard Space Flight Center
IBM	International Business Machines
I/O	input/output
JTAG	Joint Test Action Group
LED	light-emitting diode
MO&DSD	Mission Operations and Data Systems Directorate
MPU	microprocessing unit
MSB	Microelectronic Systems Branch
MUX	multiplexer
NASA	National Aeronautics and Space Administration
Nascom	NASA Communications

ABBREVIATIONS AND ACRONYMS, CONT'D

NGS	Next Generation Systems
NRZ	non-return-to-zero
OS	operating system
P&P	Plug-and-Play
PBC	PCI Bridge Chip
PCI	Peripheral Components Interface
PIFS	Parallel, Integrated FS
PLD	programmable logic device
PROM	programmable ROM
RAM	random-access memory
RLP	Return-Link Processor
ROM	read-only memory
RS	Reed-Solomon Error Detector/Corrector
SMB	sub-miniature "B"
SP	CCSDS Packet Telemetry & AOS Service Processor
SRAM	static RAM
TTL	transistor-transistor logic
VME	Versa-Module Euroboard

APPENDIX A BOARD LAYOUT AND SCHEMATICS

Appendix A includes original, unsigned, A-size top and bottom printed circuit board assembly drawings, drill master drawing, and schematic diagrams, as listed in Table A-1. Refer to Section 6 for a description of each sheet of the schematics.

Table A-1. Directory of Drawings and Diagrams

Description	Revision Level	Number of Sheets
Assembly Drawings	“ _ ”	2
Drill Master Drawing	“ _ ”	1
Schematic Diagrams	“ _ ”	13

APPENDIX B

PROGRAMMABLE DEVICE LISTINGS

This appendix includes the source file listings for all programmable devices used in the RLP, as listed in Table B-1.

Table B-1. Directory of Programmable Device Listings

Title	Issue	Date	Number of Pages
Baseboard Interface	?	?	5
DMA Interface	?	?	6
Status Collector	?	?	8
PCI Configuration ROM	?	?	1

The Baseboard Interface development directories consist of `bcon` and `bcon_1`. This is the source file, `bcon/bcon.src`.

B-2

```

ELSIF (/wr) THEN GOTO S0_1;
ELSE GOTO S1;
END IF;

STATE S2 [0001101001b]: "mem_chain_read
GOTO S3;
STATE S3 [0000101001b]:
IF (/blast) THEN
GOTO S17;
ELSE GOTO S2;
END IF;

STATE S4 [0001111011b]: "mem_chain_write
GOTO S5;
STATE S5 [0001110001b]:
GOTO S6;
STATE S6 [0000111001b]:
IF (/blast) THEN
GOTO S17;
ELSE GOTO S4;
END IF;

STATE S7 [0110111000b]: "chip_bus
IF (ack) THEN
GOTO S8;
ELSE GOTO S7;
END IF;

STATE S8 [0110011000b]: "chip_bus_en
IF (/wr) THEN
GOTO S9;
ELSE GOTO S13;
END IF;
STATE S9 [0111001001b]: "chip_bus_rd
GOTO S10;
STATE S10 [0111001011b]: "chip_bus_rd
GOTO S11;
STATE S11 [0111001111b]: "chip_bus_rd
GOTO S12;
STATE S12 [0110001101b]: "chip_bus_rd
IF (/blast) THEN
GOTO S17;
ELSE GOTO S9;
END IF;

STATE S13 [0111011011b]: "chip_bus_wr
GOTO S14;
STATE S14 [0111010011b]: "assert_we
GOTO S15;
STATE S15 [0111010111b]: "assert_we
GOTO S16;
STATE S16 [0110011111b]: "chip_bus_wr
IF (/blast) THEN
GOTO S17;
ELSE GOTO S13;
END IF;

STATE S17 [0011111011b]: "drive_ready_high
GOTO S0;

END wrcon;
END WR_gen;

PROCEDURE registers

(INPUT wr,clk,reset; INPUT addrlo[4..2]; "pins
INPUT cint; "node
INPUT overflow; "pins
INPUT osmint; "pins
INPUT osmhere; "pins
INPUT scdone; "pins
INPUT pplnff[2..0]; "pins
INPUT pplnaf[2..0]; "pins
INPUT pplnhf[2..0]; "pins
INPUT pplnae[2..0]; "pins
INPUT pplnfe[2..0]; "pins
INPUT spfaf[12..0]; "pins
INPUT spfae[12..0]; "pins
INPUT spfe[12..0]; "pins

BIPUT data[31..0] ENABLED_BY (cint*/wr); "pins

OUTPUT clrovfl CLOCKED_BY clk PRESET_BY /reset DEFAULT_TO LAST_VALUE; "node
OUTPUT control[3..2] CLOCKED_BY clk RESET_BY /reset DEFAULT_TO LAST_VALUE; "pins
OUTPUT control1 CLOCKED_BY clk PRESET_BY /reset DEFAULT_TO LAST_VALUE; "pins
OUTPUT control0 CLOCKED_BY clk RESET_BY /reset DEFAULT_TO LAST_VALUE; "pins
OUTPUT rst[16..1] CLOCKED_BY clk RESET_BY /reset DEFAULT_TO LAST_VALUE; "pins
OUTPUT rst0 CLOCKED_BY clk PRESET_BY /reset DEFAULT_TO LAST_VALUE; "pins
OUTPUT load CLOCKED_BY clk RESET_BY /reset DEFAULT_TO LAST_VALUE; "pins
OUTPUT clkssel CLOCKED_BY clk RESET_BY /reset DEFAULT_TO LAST_VALUE; "node

```

```

OUTPUT mask1[31..0]      CLOCKED_BY clk RESET_BY /reset DEFAULT_TO LAST_VALUE; "node
OUTPUT mask2[22..0]      CLOCKED_BY clk RESET_BY /reset DEFAULT_TO LAST_VALUE; "node
OUTPUT fflag1[30..0];
OUTPUT fflag2[23..0];
);
PHYSICAL NODE flagcon    CLOCKED_BY clk RESET_BY /reset DEFAULT_TO LAST_VALUE; "node

fflag1[0] = osmint;
fflag1[3..1] = pplnff[2..0];
fflag1[16..4] = spfaf[12..0];
fflag1[29..17] = spfae[12..0];
fflag1[30] = overflow;

fflag2[2..0] = pplnaf[2..0];
fflag2[5..3] = pplnhf[2..0];
fflag2[8..6] = pplnfe[2..0];
fflag2[21..9] = spfe[12..0];
fflag2[22] = scdone;
fflag2[23] = osmwhere;

CASE [cint,wr,addrlo[4..2]]

  WHEN 11000b => clrovfl = data[0]; control0 = data[1];
                  control1 = data[2]; control[3..2] = data[4..3];
                  clkssel = data[5]; load = data[6];
                  rst0 = data[7]; rst[16..1] = data[23..8];
                  flagcon = data[24];
                  "reg0_wr
  WHEN 11001b => mask1[31..0] = data[31..0];
                  "reg1_wr
  WHEN 11010b => mask2[22..0] = data[22..0];
                  "reg2_wr

END CASE;

CASE [flagcon,addrlo[4..2]]

  WHEN 0000b => data[0] = clrovfl; data[1] = control0;
                  data[2] = control1; data[4..3] = control[3..2];
                  data[5] = clkssel; data[6] = load;
                  data[7] = rst0; data[23..8] = rst[16..1];
                  data[24] = flagcon; data[31..25] = 00000000b;
                  "reg0_rd
  WHEN 1000b => data[0] = clrovfl; data[1] = control0;
                  data[2] = control1; data[4..3] = control[3..2];
                  data[5] = clkssel; data[6] = load;
                  data[7] = rst0; data[23..8] = rst[16..1];
                  data[24] = flagcon; data[31..25] = 00000000b;
                  "reg0_rd

  WHEN 0001b => data[31..0] = mask1[31..0];
                  "reg1_rd
  WHEN 1001b => data[31..0] = mask1[31..0];
                  "reg1_rd

  WHEN 0010b => data[22..0] = mask2[22..0]; data[31..23] = 000000000b;
                  "reg2_rd
  WHEN 1010b => data[22..0] = mask2[22..0]; data[31..23] = 000000000b;
                  "reg2_rd

  WHEN 0011b => data[30..0] = fflag1[30..0]; data[31] = 0b;
                  "reg3_rd_f
  WHEN 1011b => data[16..0] = fflag1[16..0] + /mask1[16..0];
                  data[30..17] = fflag1[30..17] * mask1[30..17];
                  data[31] = 0b;
                  "reg3_rd_mf

  WHEN 0100b => data[23..0] = fflag2[23..0]; data[31..24] = 00000000b;
                  "reg4_rd_f
  WHEN 1100b => data[5..0] = fflag2[5..0] + /mask2[5..0];
                  data[22..8] = fflag2[22..8] * mask2[22..8];
                  data[23] = fflag2[23]; data[31..24] = 00000000b;
                  "reg4_rd_mf

ELSE data[31..0] = 00000000h;
END CASE;

END registers;

PROCEDURE clk_mux

(INPUT intclk;
 INPUT extclk;
 INPUT clkssel;
 OUTPUT refclkout;
);
"pins
"pins
"node
"pins

refclkout = (intclk * /clkssel) + (extclk * clkssel);

END clk_mux;

PROCEDURE over_flow

(INPUT ovfwrclk;
 INPUT reset;
 INPUT ovfwe;
 INPUT ovfff;
 INPUT clrovfl;
 OUTPUT overflow
);
"pins
"pins
"pins
"node
"node
"node

CLOCKED_BY ovfwrclk RESET_BY /reset;

```

```

IF (clrovfl) THEN
    overflow = 0b;
ELSIF (/overflow * /ovfwe * /ovfff) THEN
    overflow = 1b;
ELSE
    overflow = overflow;
END IF;

END over_flow;

PROCEDURE interrupt_gen

(INPUT mask1[31..0];
 INPUT mask2[22..0];
 INPUT fflag1[30..0];
 INPUT fflag2[23..0];

    "node
    "node
    "node
    "node

    OUTPUT interrupt;
    "OUTPUT interrupt ENABLED_BY /intcon;
    );
    PHYSICAL NODE intmask[10..0];
    "pins
    "node

    "interrupt = 0b;

intmask[10] = mask2[22] * fflag2[22]
+ mask2[21] * fflag2[21]
+ mask2[20] * fflag2[20]
+ mask2[19] * fflag2[19];
intmask[9] = mask2[18] * fflag2[18]
+ mask2[17] * fflag2[17]
+ mask2[16] * fflag2[16]
+ mask2[15] * fflag2[15]
+ mask2[14] * fflag2[14];
intmask[8] = mask2[13] * fflag2[13]
+ mask2[12] * fflag2[12]
+ mask2[11] * fflag2[11]
+ mask2[10] * fflag2[10]
+ mask2[9] * fflag2[9];
intmask[7] = mask2[8] * fflag2[8]
+ mask2[7] * fflag2[7]
+ mask2[6] * fflag2[6]
+ mask2[5] * fflag2[5]
+ mask2[4] * fflag2[4];
intmask[6] = mask2[3] * fflag2[3]
+ mask2[2] * fflag2[2]
+ mask2[1] * fflag2[1]
+ mask2[0] * fflag2[0]
+ mask1[30] * fflag1[30];
intmask[5] = mask1[29] * fflag1[29]
+ mask1[28] * fflag1[28]
+ mask1[27] * fflag1[27]
+ mask1[26] * fflag1[26]
+ mask1[25] * fflag1[25];
intmask[4] = mask1[24] * fflag1[24]
+ mask1[23] * fflag1[23]
+ mask1[22] * fflag1[22]
+ mask1[21] * fflag1[21]
+ mask1[20] * fflag1[20];
intmask[3] = mask1[19] * fflag1[19]
+ mask1[18] * fflag1[18]
+ mask1[17] * fflag1[17]
+ mask1[16] * fflag1[16]
+ mask1[15] * fflag1[15];
intmask[2] = mask1[14] * fflag1[14]
+ mask1[13] * fflag1[13]
+ mask1[12] * fflag1[12]
+ mask1[11] * fflag1[11]
+ mask1[10] * fflag1[10];
intmask[1] = mask1[9] * fflag1[9]
+ mask1[8] * fflag1[8]
+ mask1[7] * fflag1[7]
+ mask1[6] * fflag1[6]
+ mask1[5] * fflag1[5];
intmask[0] = mask1[4] * fflag1[4]
+ mask1[3] * fflag1[3]
+ mask1[2] * fflag1[2]
+ mask1[1] * fflag1[1]
+ mask1[0] * fflag1[0] * fflag2[23];

IF (((intmask[10..0]) = 000h) OR (mask1[31]=0b)) THEN
    interrupt = 1b;
ELSE interrupt = 0b;
END IF;

END interrupt_gen;

OUTPUT spfaf0t3, spfaf4t11;
    "pins

INPUT addr[27..18];
INPUT addrlo[4..2];
    "pins
    "pins

```

```

INPUT clk;                                "pins
INPUT ads;                                "pins
INPUT blast;                              "pins
INPUT wr;                                 "pins
INPUT ack;                                "pins
INPUT reset;                              "pins

INPUT pplnff[2..0];                       "pins
INPUT pplnaf[2..0];                       "pins
INPUT pplnhf[2..0];                       "pins
INPUT pplnae[2..0];                       "pins
INPUT pplnfe[2..0];                       "pins
INPUT spfaf[12..0];                       "pins
INPUT spfae[12..0];                       "pins
INPUT spfe[12..0];                       "pins
INPUT intclk;                             "pins
INPUT extclk;                             "pins

INPUT ovfwrclk;                           "pins
INPUT ovfwe;                              "pins
INPUT ovfff;                              "pins

INPUT osmint;                             "pins
INPUT osmhere;                            "pins
INPUT scdone;                             "pins

BIPUT data[31..0];                        "pins

OUTPUT xen;                               "pins
OUTPUT oe;                                "pins
OUTPUT we;                                "pins
OUTPUT memcs;                             "pins
OUTPUT bready;                            "pins
OUTPUT hold;                              "pins

OUTPUT control[3..2];                     "pins
OUTPUT control1;                          "pins
OUTPUT control0;                          "pins
OUTPUT rst[16..1];                        "pins
OUTPUT rst0;                              "pins
OUTPUT load;                              "pins

OUTPUT refclkout;                         "pins

OUTPUT interrupt;                         "pins

NODE overflow;
NODE cint;

NODE clkssel;
NODE clrovfl;
NODE mask1[31..0];
NODE mask2[22..0];
NODE fflag1[30..0];
NODE fflag2[23..0];

SP_hold      (spfaf[11..0], spfaf0t3, spfaf4t11);
WR_gen       (addr[27..18], clk, ads, blast, wr, ack, reset, rst[10],
              cint, xen, oe, we, memcs, bready, hold);

registers    (wr, clk, reset, addrlo[4..2], cint, overflow, osmint, osmhere,
              scdone, pplnff[2..0], pplnaf[2..0], pplnhf[2..0],
              pplnfe[2..0], spfaf[12..0], spfae[12..0], spfe[12..0],
              data[31..0], clrovfl, control[3..2], control1, control0,
              rst[16..1], rst0, load, clkssel,
              mask1[31..0], mask2[22..0], fflag1[30..0], fflag2[23..0]);

clk_mux      (intclk, extclk, clkssel, refclkout);

over_flow    (ovfwrclk, reset, ovfwe, ovfff, clrovfl, overflow);

interrupt_gen(mask1[31..0], mask2[22..0], fflag1[30..0], fflag2[23..0],
              interrupt);

```

B.2 DMA INTERFACE PROGRAM LISTING

The DMA Interface development directories consist of dmai and dmai_1. This is the source file, dmai/dmai.src.

```
#TITLE 'DMA Interface';
#ENGINEER 'Andy Wolf';
#REVISION '-';
#PROJECT 'NextGen Return Link Processor Card';
#COMMENT '';

" REVISION HISTORY
" 8/7/96   Created rev -.
"

PROCEDURE SP_fifo
(
  INPUT fclk, tripins, reset, spemptyn[12..0];
  INPUT spfifo[3..0], oe, rd;
  OUTPUT spempty;
  OUTPUT oen[12..0] ENABLED_BY /tripins CLOCKED_BY fclk PRESET_BY reset DEFAULT_TO 01FFFh; "pins
  OUTPUT rdn[12..0] ENABLED_BY /tripins CLOCKED_BY fclk PRESET_BY reset DEFAULT_TO 01FFFh; "pins
);

CASE spfifo[3..0]
  WHEN 0 => oen[0] = /oe; rdn[0] = /rd; spempty = /spemptyn[0];
  WHEN 1 => oen[1] = /oe; rdn[1] = /rd; spempty = /spemptyn[1];
  WHEN 2 => oen[2] = /oe; rdn[2] = /rd; spempty = /spemptyn[2];
  WHEN 3 => oen[3] = /oe; rdn[3] = /rd; spempty = /spemptyn[3];
  WHEN 4 => oen[4] = /oe; rdn[4] = /rd; spempty = /spemptyn[4];
  WHEN 5 => oen[5] = /oe; rdn[5] = /rd; spempty = /spemptyn[5];
  WHEN 6 => oen[6] = /oe; rdn[6] = /rd; spempty = /spemptyn[6];
  WHEN 7 => oen[7] = /oe; rdn[7] = /rd; spempty = /spemptyn[7];
  WHEN 8 => oen[8] = /oe; rdn[8] = /rd; spempty = /spemptyn[8];
  WHEN 9 => oen[9] = /oe; rdn[9] = /rd; spempty = /spemptyn[9];
  WHEN 10 => oen[10] = /oe; rdn[10] = /rd; spempty = /spemptyn[10];
  WHEN 11 => oen[11] = /oe; rdn[11] = /rd; spempty = /spemptyn[11];
  WHEN 12 => oen[12] = /oe; rdn[12] = /rd; spempty = /spemptyn[12];
END CASE;

END SP_fifo;

PROCEDURE dma_control
(
  INPUT fclk, reset;
  INPUT fclk, reset, tripins;
  INPUT cnta[16..0], cntb[16..0], dmfull;
  OUTPUT go, allin;
  OUTPUT cntenaba, cntenabb;
  OUTPUT channela, channelb;
);

VIRTUAL NODE cnteq0a, cnteq0b, cnteq1a, cnteq1b;
PHYSICAL NODE qcnteq0a, qcnteq0b;
PHYSICAL NODE goa, gob;

"detect cnters equal 1 or 0
IF (cnta = 1) THEN cnteq1a = 1; else cnteq1a = 0; END IF;
IF (cnta = 0) THEN cnteq0a = 1; else cnteq0a = 0; END IF;
IF (cntb = 1) THEN cnteq1b = 1; else cnteq1b = 0; END IF;
IF (cntb = 0) THEN cnteq0b = 1; else cnteq0b = 0; END IF;

"select which DMA channel is active
qcnteq0a.d = cnteq0a;
qcnteq0b.d = cnteq0b;
"channela.d = /qcnteq0a * (qcnteq0b + /channelb);
"channelb.d = /qcnteq0b * (qcnteq0a + /channela);
channela.d = /qcnteq0a * (qcnteq0b + /channelb) * /tripins;
channelb.d = /qcnteq0b * (qcnteq0a + /channela) * /tripins;
allin.d = qcnteq0a * qcnteq0b;

"control signals enable cnters and start other actions
cntenaba.d = channela * /cnteq0a * /(cnteq1a * cntenaba) * /dmfull;
goa.d = channela * /cnteq0a * /(cnteq1a * cntenaba);

cntenabb.d = channelb * /cnteq0b * /(cnteq1b * cntenabb) * /dmfull;
```

```
gob.d      = channelb * /cnteq0b * /(cnteq1b * cntenabb);
```

```
IF channelb THEN go.d = gob;
ELSE           go.d = goa;
END IF;
```

```
END dma_control;
```

PROCEDURE registers

```
(INPUT fclk, reset, fifodata[8..0];           "pins
INPUT grabdata, badread, dataenab;           "nodes
INPUT cntaspace, cntbspace, garbpacea, garbpaceb; "nodes
INPUT setupspace, SPreadspace;               "nodes
INPUT cntenaba, cntenabb, channela, channelb, load, loadc; "nodes
OUTPUT spfifo[3..0];                          "nodes
OUTPUT byteorder CLOCKED_BY fclk RESET_BY reset DEFAULT_TO LAST_VALUE;
OUTPUT dmrstn   CLOCKED_BY fclk PRESET_BY reset DEFAULT_TO LAST_VALUE;
OUTPUT cnta[16..0], cntb[16..0] CLOCKED_BY fclk RESET_BY reset DEFAULT_TO LAST_VALUE;
BIPUT  data[31..0] ENABLED_BY dataenab CLOCKED_BY fclk RESET_BY reset DEFAULT_TO LAST_VALUE;
);
```

```
PHYSICAL NODE spfifoa [ 3..0] CLOCKED_BY fclk RESET_BY reset DEFAULT_TO LAST_VALUE;
PHYSICAL NODE spfifob [ 3..0] CLOCKED_BY fclk RESET_BY reset DEFAULT_TO LAST_VALUE;
PHYSICAL NODE garbagea [16..0] CLOCKED_BY fclk RESET_BY reset DEFAULT_TO LAST_VALUE;
PHYSICAL NODE garbageb [16..0] CLOCKED_BY fclk RESET_BY reset DEFAULT_TO LAST_VALUE;
VIRTUAL NODE data_comb[20..0] DEFAULT_TO 0;
PHYSICAL NODE loada, loadb CLOCKED_BY fclk RESET_BY reset;
PHYSICAL NODE grabdata1, grabdata2, grabdata3 CLOCKED_BY fclk RESET_BY reset;
PHYSICAL NODE garbcnta, garbcntb CLOCKED_BY fclk RESET_BY reset;
```

"Control Registers

```
loada.d = loadc * cntaspace;
loadb.d = loadc * cntbpace;
```

```
IF (loada)      THEN cnta=data[16..0]; spfifoa = data[20..17];
ELSIF (cntenaba) THEN cnta = cnta .-. 1;
END IF;
```

```
IF (loadb)      THEN cntb=data[16..0]; spfifob = data[20..17];
ELSIF (cntenabb) THEN cntb = cntb .-. 1;
END IF;
```

```
IF channelb THEN spfifo = spfifob;
ELSE          spfifo = spfifoa;
END IF;
```

"Garbage Registers

```
garbcnta.d = badread * /garbcntb * (channela + garbcnta);
garbcntb.d = badread * /garbcnta * (channelb + garbcntb);
```

```
IF loada THEN garbagea = 0;
ELSIF garbcnta THEN garbagea = garbagea .+. 1;
END IF;
```

```
IF loadb THEN garbageb = 0;
ELSIF garbcntb THEN garbageb = garbageb .+. 1;
END IF;
```

"Setup Register

```
"IF (load * setupspace) THEN byteorder.d = data[0]; dmrstn.d = data[1];
IF (load * setupspace) THEN byteorder = data[0]; dmrstn = data[1];
END IF;
```

"Output Data Multiplexor

```
IF cntaspace THEN data_comb[16..0]= cnta; data_comb[20..17]=spfifoa;
ELSIF cntbpace THEN data_comb[16..0]= cntb; data_comb[20..17]=spfifob;
ELSIF garbpacea THEN data_comb[16..0]= garbagea;
ELSIF garbpaceb THEN data_comb[16..0]= garbageb;
ELSIF setupspace THEN data_comb[0] = byteorder; data_comb[1] = dmrstn;
ELSIF SPreadspace THEN data_comb[ 8..0]= fifodata;
END IF;
```

```
"Grab data into output register synchronously with Fclk
grabdata1 = grabdata; "delay needed for SPread to get data from FIFOs
grabdata2 = grabdata1;
grabdata3 = grabdata2;
data[31..21] = 0;
IF (grabdata3) THEN data[20..0] = data_comb; END IF;
```



```
END registers;
```

```
PROCEDURE memory_map
```

```
(INPUT addr[20..17],sel[4..2], lclk,reset;
OUTPUT dataspace DEFAULT_TO 0;
OUTPUT dmaspace,SPreadspace,dmwwritespace DEFAULT_TO 0;
OUTPUT cntaspace,cntbspace,garbpacea,garbpaceb,setupspace DEFAULT_TO 0;
);
```

```
VIRTUAL NODE regspace DEFAULT_TO 0;
VIRTUAL NODE SPspace DEFAULT_TO 0;
VIRTUAL NODE DWspace DEFAULT_TO 0;
```

```
IF (addr[20..17] = 4) THEN dmaspace = 1;
ELSIF (addr[20..17] = 5) THEN regspace = 1;
ELSIF (addr[20..17] = 6) THEN SPspace = 1;
ELSIF (addr[20..17] = 7) THEN DWspace = 1;
END IF;
```

```
IF (regspace AND (sel=0)) THEN cntaspace = 1;
ELSIF (regspace AND (sel=1)) THEN cntbspace = 1;
ELSIF (regspace AND (sel=2)) THEN garbpacea = 1;
ELSIF (regspace AND (sel=3)) THEN garbpaceb = 1;
ELSIF (regspace AND (sel=4)) THEN setupspace = 1;
END IF;
```

```
SPreadspace = SPspace; "SPspace is virtual node, SPreadspace is physical node
dmwwritespace = DWspace;
dataspace=regspace+SPspace+DWspace;
```

```
END memory_map;
```

```
PROCEDURE control_logic
```

```
(INPUT fclk,reset,dmafn; "pins
INPUT go,spempty,wrcnteq3, SPreadspace, grabdata; "nodes
OUTPUT rd, oe, we CLOCKED_BY fclk RESET_BY reset; "nodes
OUTPUT wrntclrn,badread, dmfull CLOCKED_BY fclk RESET_BY reset; "nodes
);
```

```
PHYSICAL NODE full, full1, full2 CLOCKED_BY fclk RESET_BY reset;
PHYSICAL NODE goq,goqq CLOCKED_BY fclk RESET_BY reset;
VIRTUAL NODE SPread;
PHYSICAL NODE SPreadq CLOCKED_BY fclk RESET_BY reset;
```

```
rd = go * /spempty * /badread * /full1 + SPread;
oe = go + goq + SPread + SPreadq;
we = goq * /full2;
wrcntclrn.d = go * goq * /full2;
badread.d = goqq * (spempty + badread);
dmfull.d = /dmafn * (wrcnteq3 + dmfull);
full.d = dmfull;
full1.d = full;
full2.d = full1;
goq.d = go;
goqq.d = goq;
SPread = SPreadspace * grabdata;
SPreadq.d = SPread;
```

```
END control_logic;
```

```
PROCEDURE bus_access
```

```
("(INPUT fclk,tripins,reset,lclk,adsn,write,blastn,dmaen; "pins
(INPUT fclk,reset,lclk,adsn,write,blastn,dmaen; "pins
INPUT dataspace, dmaspace, allin, dmempty; "nodes
OUTPUT dataenab; "output enables
OUTPUT regaccess CLOCKED_BY lclk RESET_BY reset; "nodes
OUTPUT load CLOCKED_BY fclk RESET_BY reset; "nodes
OUTPUT loadc; "nodes
OUTPUT grabdata; "nodes
OUTPUT grabdata CLOCKED_BY fclk RESET_BY reset; "nodes
OUTPUT dmrndn ENABLED_BY /tripins; "pins
OUTPUT dmoen ENABLED_BY /tripins CLOCKED_BY lclk PRESET_BY reset DEFAULT_TO 1; "pins
OUTPUT dmrndn; "pins
OUTPUT dmoen CLOCKED_BY lclk PRESET_BY reset DEFAULT_TO 1;
OUTPUT readyn ENABLED_BY readynab;
);
```

```

"PHYSICAL NODE fcnt[2..0] CLOCKED_BY fclk RESET_BY reset DEFAULT_TO LAST_VALUE;
PHYSICAL NODE fcnt[1..0] CLOCKED_BY fclk RESET_BY reset DEFAULT_TO LAST_VALUE;
PHYSICAL NODE regaccessf CLOCKED_BY fclk RESET_BY reset;
PHYSICAL NODE fdone CLOCKED_BY fclk RESET_BY reset DEFAULT_TO LAST_VALUE;
PHYSICAL NODE access, done CLOCKED_BY lclk RESET_BY reset;
"VIRTUAL NODE fcntrst;

PHYSICAL NODE regaccessl CLOCKED_BY lclk RESET_BY reset;
PHYSICAL NODE ready CLOCKED_BY lclk RESET_BY reset;
PHYSICAL NODE rdff CLOCKED_BY lclk RESET_BY reset;
PHYSICAL NODE readyenab CLOCKED_BY lclk RESET_BY reset; "output enable
VIRTUAL NODE regready;
VIRTUAL NODE dmready,dmaaccess,endaccess,enddmard;

"register access

regaccess.d = /adsn*datspace + blastn*ready*datspace + regaccess*/done;
regaccessf.d = regaccess;
regaccessl.d = regaccess;
"done.d = maxcount;
"done.d = maxcount * /fcntrst;
done.d = fdone * regaccess;
"fcntrst = /regaccess + /regaccessf;
regready = done * regaccess;
dataenab = /write * regaccessl;

IF /regaccessf THEN fcnt = 0;
ELSIF /fdone THEN fcnt = fcnt .+. 1; END IF;
"ELSIF /maxcount THEN fcnt = fcnt .+. 1; END IF;

"IF (fcnt = 5) THEN maxcount = 1; ELSE maxcount = 0; END IF;
IF ((fcnt >= 2) AND regaccessf) THEN fdone = 1; ELSE fdone = 0; END IF;
IF ( write AND (fcnt = 1)) THEN loadc = 1; ELSE loadc = 0; END IF;
IF (/write AND (fcnt = 1)) THEN grabdata = 1; ELSE grabdata = 0; END IF;
load.d = loadc; "load is clocked output, loadc is combinatorial output

"dma access

enddmard = /blastn * ready;
endaccess = /blastn * /ready;
access.d = /adsn * dmspace + access * /endaccess;
dmaaccess = access * /write;
dmoen.d = /(dmaaccess * /endaccess);
"rdff.d = dmaaccess * /endaccess * (allin + dmaen + rdff);
rdff.d = dmaaccess * /endaccess * /dmempty * (allin + dmaen + rdff);
"dmrdn = /(rdff * /endaccess);
dmrdn = /(rdff * /dmempty * /enddmard);
"dmready = rdff * /endaccess;
dmready = rdff * /dmempty * /enddmard;

"common to both dma or register access
ready.d = regready + dmready;
"readyenab.d = ((regready + dmready) + ready) * /tripins;
readyenab.d = ((regready + dmready) + ready);
readyn = /ready;
END bus_access;

PROCEDURE DMAI_fifo
(
  INPUT fclk, tripins, reset, data[31..0]; "pins
  INPUT wrntclrn, we, dmwritespace, load, write, oe, go, regaccess, byteorder; "nodes
  OUTPUT wrnteq3; "nodes
  OUTPUT fifodata[8..0] ENABLED_BY fifodataenab; "pins
  "OUTPUT wen[3..0] ENABLED_BY /tripins CLOCKED_BY fclk PRESET_BY reset DEFAULT_TO 0Fh;
  OUTPUT wen[3..0] CLOCKED_BY fclk PRESET_BY reset DEFAULT_TO 0Fh;
);

PHYSICAL NODE wrnt[1..0] CLOCKED_BY fclk RESET_BY reset;
VIRTUAL NODE dmfifowrite;
PHYSICAL NODE fifodataenab; "output enable
VIRTUAL NODE nocontention;

nocontention = /oe * /go;
"fifodataenab = dmwritespace * write * access * nocontention * /tripins;
fifodataenab = dmwritespace * write * regaccess * nocontention * /tripins;
fifodata[8..0] = data[8..0];

dmfifowrite = load * dmwritespace ;

IF wrntclrn THEN wrnt = wrnt .+. 1; ELSE wrnt = 0; END IF;
IF (wrnt=3) THEN wrnteq3 = 1; ELSE wrnteq3 = 0; END IF;

IF dmfifowrite THEN wen[3..0] = 0;
ELSIF byteorder THEN CASE wrnt
  WHEN 0 => wen[3] = /we;

```

```

        WHEN 1 => wen[2] = /we;
        WHEN 2 => wen[1] = /we;
        WHEN 3 => wen[0] = /we;
    END CASE;
ELSE
    CASE wrcnt
        WHEN 0 => wen[0] = /we;
        WHEN 1 => wen[1] = /we;
        WHEN 2 => wen[2] = /we;
        WHEN 3 => wen[3] = /we;
    END CASE;
END IF;

END DMAI_fifo;

INPUT    lclk;                "clock for i960 bus - typically 33MHz
INPUT    fclk;                "FIFO clock - typically 56-66MHz

INPUT    addr[20..17],sel[4..2]; "i960 bus
INPUT    adsn,blastn;         "active low i960 bus signal
INPUT    write;               "1 = write, 0 = read i960 bus signal
"INPUT    hold;                "1 = PBC requests bus

INPUT    spemptyn[12..0];     "active low empty flags from SP FIFOs
INPUT    dmhalfn, dmfulln ;   "active low flags from DMAI FIFO
INPUT    dmaen,dmafn;         "active low programmable flags from DMAI FIFO
INPUT    dmen0, dmen3;        "active low empty flags from DMAI FIFOs

INPUT    tripins, resetn;     "set DMAI mode of operation

"i960 bus bidirectional signals
BIPUT    data[31..0],readyn;
BIPUT    btermn ENABLED_BY 0; "i960 bus burst termination - unused

"FIFO data bus bidirectional signals
BIPUT    fifodata[8..0];

"Bus arbitration
"BIPUT    holda CLOCKED_BY lclk RESET_BY reset ENABLED_BY /tripins; "1 = PBC is granted access to bus

"active low SP FIFO enab
OUTPUT    oen[12..0], rdh[12..0];

"active low DMAIFIFO write
OUTPUT    wen[3..0];

"active low DMAIFIFO read
OUTPUT    dmoen, dmrhn;
OUTPUT    dmrstn;             "active low reset for DMAI FIFO

VIRTUAL  NODE cntenaba,cntenabb; "dma control FIFO nodes
VIRTUAL  NODE channela,channelb; "dma control nodes
VIRTUAL  NODE cnta[16..0], cntb[16..0]; "register bits
VIRTUAL  NODE byteorder;         "register bits
VIRTUAL  NODE wrcntclrn,we,wrcnteq3; "DMAI FIFO nodes
VIRTUAL  NODE oe,rd;             "SP FIFO nodes
PHYSICAL NODE spfifo[3..0];      "SP FIFO nodes
VIRTUAL  NODE spempty;           "SP FIFO nodes
VIRTUAL  NODE load, loadc, regaccess; "register access nodes
VIRTUAL  NODE dmfull, badread, go; "control sigs
VIRTUAL  NODE grabdata,allin;    "control sigs
PHYSICAL NODE dataenab;          "output enable sigs
VIRTUAL  NODE dmaspace, dataspace; "memory map
PHYSICAL NODE dmwritespace;      "memory map
PHYSICAL NODE cntaspace,cntbpace; "memory map
PHYSICAL NODE garbpspacea,garbpspaceb; "memory map
PHYSICAL NODE setupspace,SPreadspace; "memory map
VIRTUAL  NODE reset;            "global signals
VIRTUAL  NODE dmempty;           "to bus access

reset = /resetn;
"holda = hold;
btermn = 0;
dmempty = /byteorder * /dmen3 + byteorder * /dmen0;

SP_fifo (fclk, tripins, reset, spemptyn[12..0],spfifo[3..0], oe, rd,
        spempty, oen[12..0], rdh[12..0]);

"dma_control (fclk,reset,cnta[16..0],cntb[16..0],dmfull,
dma_control (fclk,reset,tripins,cnta[16..0],cntb[16..0],dmfull,
        go,allin,cntenaba,cntenabb,channela,channelb);

```

```

registers (fclk,reset,fifodata[8..0],grabdata,badread,dataenab,
          cntaspace, cntbspace,garbspacea,garbspaceb,setupspace,SPreadspace,
          cntenaba, cntenabb, channela, channelb, load, loadc,
          spfifo[3..0],byteorder,dmrstn,cnta[16..0],cntb[16..0],data[31..0]);

memory_map (addr[20..17],sel[4..2], lclk, reset,
          dataspace,dmaspace,SPreadspace,dmwritespace,
          cntaspace,cntbspace,garbspacea,garbspaceb,setupspace);

control_logic (fclk,reset,dmafn,go,spempty,wrcnteq3,SPreadspace,grabdata,
          rd, oe, we, wrntclrn,badread, dmfull);

"bus_access (fclk,tripins,reset,lclk,adsn,write,blastn,
bus_access (fclk,reset,lclk,adsn,write,blastn,
          dmaen,dataspace,dmaspace, allin, dmempty,
          dataenab, regaccess, load, loadc, grabdata, dmrnd, dmoen, readyn);

DMAI_fifo (fclk,tripins,reset,data[31..0],wrcntclrn,we,dmwritespace,load,
          write,oe,go,regaccess,byteorder,wrcnteq3,fifodata[8..0], wen[3..0]);

```

B.3 STATUS COLLECTOR PROGRAM LISTING

The Status Collector development directories consist of stcol and stcol_1. This is the source file, stcol/stcol.src.

```

#TITLE      'Status collect chip for Return Link Processor card';
#ENGINEER   'Fred Peng';
#REVISION   '-';
#COMMENT     'This design collects the status from several devices'
            'including PIFS chip, RSEC chip, SP chip, A/D etc.'
            'and then store them in the 9 bit Synchronous FIFO'
            'eod is the mark bit to indicate the last byte of'
            'every status piece';

INPUT       we, hold, aff,clk, clr;
INPUT       adcd[8];
BIPUT       addr[18] ENABLED_BY out_ena;
BIPUT       data[32] ENABLED_BY out_en;
BIPUT       rdb CLOCKED_BY clk PRESET_BY /clr ENABLED_BY out_ena DEFAULT_TO 1b;
OUTPUT      csb[7];
PHYSICAL NODE div[4] CLOCKED_BY clk RESET_BY /clr;
PHYSICAL NODE adcreg[16] CLOCKED_BY adck RESET_BY /clr DEFAULT_TO LAST_VALUE;
OUTPUT      adck;
OUTPUT      adccs CLOCKED_BY adck PRESET_BY /clr;
OUTPUT      adca[2] CLOCKED_BY adck RESET_BY /clr;
OUTPUT      wenb;
OUTPUT      sc_done CLOCKED_BY clk RESET_BY /clr DEFAULT_TO 0b;
OUTPUT      ack CLOCKED_BY clk;
NODE        sb[9] CLOCKED_BY clk;
NODE        xbl CLOCKED_BY clk;
NODE        cnt5_en, cnt3_en CLOCKED_BY clk RESET_BY /clr DEFAULT_TO 0b;
NODE        cnt2_en, cnt3a_en CLOCKED_BY clk RESET_BY /clr DEFAULT_TO 0b;
NODE        clr_addr;
NODE        cnt5[3] CLOCKED_BY clk;
NODE        cnt3[3] CLOCKED_BY clk;
NODE        cnt2[2] CLOCKED_BY clk;
NODE        cnt3a[3] CLOCKED_BY clk;
NODE        cnt1[1] CLOCKED_BY clk;
NODE        addro[16] CLOCKED_BY clk RESET_BY /clr;
NODE        latch_dat[32] CLOCKED_BY clk RESET_BY /clr;
NODE        ctrlreg[27] CLOCKED_BY clk RESET_BY /clr;
NODE        dat_sel CLOCKED_BY clk RESET_BY /clr DEFAULT_TO 0b;
NODE        out_sel[2] CLOCKED_BY clk RESET_BY /clr DEFAULT_TO 00b;
NODE        muxout[32];
NODE        fr_addr[14];
NODE        pk_addr[14];
NODE        out_en CLOCKED_BY clk RESET_BY /clr DEFAULT_TO 0b;
NODE        addr_en DEFAULT_TO 0b;
NODE        cs[5] CLOCKED_BY clk PRESET_BY /clr DEFAULT_TO 11111b;
"PHYSICAL NODE clr_addr1,clr_addr2,eod1,eod2 DEFAULT_TO 0b;
PHYSICAL NODE clr_addr1 DEFAULT_TO 0b;
PHYSICAL NODE clr_addr3 CLOCKED_BY clk RESET_BY /clr DEFAULT_TO 0b;
PHYSICAL NODE cs_a[7] DEFAULT_TO 1111111b;
PHYSICAL NODE out_ena;
PHYSICAL NODE cs_sc DEFAULT_TO 1b;
PHYSICAL NODE cs_fifo DEFAULT_TO 1b;
PHYSICAL NODE ack1 CLOCKED_BY clk RESET_BY /clr DEFAULT_TO 0b;
"PHYSICAL NODE frm_on;
"PHYSICAL NODE pkt_on;
PHYSICAL NODE fr_eq CLOCKED_BY clk;
PHYSICAL NODE pk_eq CLOCKED_BY clk;
PHYSICAL NODE clr_cnt2,clr_cnt3,clr_cnt5 DEFAULT_TO 0b;
PHYSICAL NODE clr_cnt3a CLOCKED_BY clk RESET_BY /clr DEFAULT_TO 0b;
PHYSICAL NODE wen CLOCKED_BY clk RESET_BY /clr DEFAULT_TO 0b;
PHYSICAL NODE eod CLOCKED_BY clk RESET_BY /clr DEFAULT_TO 0b;
PHYSICAL NODE fsdone,rsdone,spdone DEFAULT_TO 0b;

```

```

MACRO gen_cs0
{
  CASE cnt5
    WHEN 0..1 =>
      cs[0] = 0b;
    ELSE
      cs[0] = 1b;
    END CASE;
}
MACRO gen_cs(y)
{
  CASE cnt3
    WHEN 0..1 =>
      cs[y] = 0b;
    ELSE
      cs[y] = 1b;
    END CASE;
}
MACRO gen_cs3
{
  CASE cnt2
    WHEN 0..1 =>
      cs[3] = 0b;
    ELSE
      cs[3] = 1b;
    END CASE;
}

MACRO gen_cs4
{
  CASE cnt5
    WHEN 0..1 =>
      cs[4] = 0b;
    ELSE
      cs[4] = 1b;
    END CASE;
}

" MACRO gen_cs5
" {
"   CASE cnt2
"     WHEN 0..1 =>
"       cs[5] = 0b;
"     ELSE
"       cs[5] = 1b;
"     END CASE;
" }

MACRO out_1byte
{
  CASE cnt2
    WHEN 0..1 => rdb = 0b;
    WHEN 2 => [out_sel,wen,out_en,addr_en] = 00111b;
    END CASE;
}

MACRO out_2byte
{
  CASE cnt3
    WHEN 0..1 => rdb = 0b;
    WHEN 2 => [out_sel,wen,out_en] = 0111b;
    WHEN 3 => [out_sel,wen,out_en,addr_en] = 00111b;
    END CASE;
}

MACRO out_4byte
{
  CASE cnt5
    WHEN 0..1 => rdb = 0b;
    WHEN 2 => [out_sel,wen,out_en] = 1111b;
    WHEN 3 => [out_sel,wen,out_en] = 1011b;
    WHEN 4 => [out_sel,wen,out_en] = 0111b;
    WHEN 5 => [out_sel,wen,out_en,addr_en] = 00111b;
    END CASE;
}

MACRO ctrlreg_out
{
  CASE cnt3a
    WHEN 000b => out_sel = 11b;
    WHEN 001b => out_sel = 10b;
    WHEN 010b => out_sel = 01b;
    WHEN 011b => out_sel = 00b;
    WHEN 100b => out_sel = 11b;addr_en = 1b;
    WHEN 101b => out_sel = 10b;
    WHEN 110b => out_sel = 01b;
    WHEN 111b => out_sel = 00b;
    END CASE;
}

" 32 bit input data latch"

```

```

IF      (rdb = 0b) THEN
    latch_dat[31..0] = data[31..0];
ELSE
    latch_dat[31..0] = latch_dat[31..0];
END IF;

" 32bit wide 2 to 1 mux
" when collecting status on ctrlreg and adcreg, data[8] will
" not represent eod; instead it will be ctrlreg[8] or adcreg[8]

IF dat_sel THEN
    IF (addr[0] = 0b) THEN
        muxout[26..0] = ctrlreg[26..0];
        muxout[31..27] = 00000b;
        data[8] = ctrlreg[8];
    ELSE
        muxout[15..0] = adcreg[15..0];
        muxout[31..16] = 0000000000000000b;
        data[8] = adcreg[8];
    END IF;
ELSE
    muxout[31..0] = latch_dat[31..0];
    data[8] = eod;
END IF;

" Byte-wide 4 to 1 Mux data output"

CASE out_sel
    WHEN 00b => data[7..0] = muxout[7..0];
    WHEN 01b => data[7..0] = muxout[15..8];
    WHEN 10b => data[7..0] = muxout[23..16];
    WHEN 11b => data[7..0] = muxout[31..24];
END CASE;

"
"     CASE sb
"         WHEN 0 => sc_done = 1b;
"         WHEN 256..262 => ack1 = 1b;
"     END CASE;

IF sb = 0 THEN sc_done = 1b; ELSE sc_done = 0b; END IF;
ack1 = sb[8];

" Muxout chip selects for bus access or status collect

IF (ack) THEN
    csb[6..0] = cs_a[6..0];
    wenb = /cnt1;
    out_ena = 0b;
ELSE
    csb[4..0] = cs[4..0];
    csb[6..5] = 11b;
    wenb = /wen;
    out_ena = 1b;
END IF;

" Check if need to collect frame status

"IF (ctrlreg[10..3] <> 0) THEN
"    frm_on = 1b;
"    ELSE
"        frm_on = 0b;
"END IF;

" Check if need to collect packet status

"IF (ctrlreg[24..11] <> 0) THEN
"    pkt_on = 1b;
"    ELSE
"        pkt_on = 0b;
"END IF;

" Check if finish frame status collect

"IF ((addro[11..0] = fr_addr[11..0]) AND (addr_en)) THEN
IF (addro[11..0] = fr_addr[11..0]) THEN
    fr_eq = 1b;
    ELSE
        fr_eq = 0b;
END IF;

" Check if finish packet status collect

"IF ((addro[13..0] = pk_addr[13..0]) AND (addr_en)) THEN
IF (addro[13..0] = pk_addr[13..0]) THEN
    pk_eq = 0b;
    ELSE
        pk_eq = 1b;
END IF;

IF (addro[4..0] = 11111b) THEN
    fsdone = 1b;

```

```

ELSE
    fsdone = 0b;
END IF;

IF (addro[5..0] = 100111b) THEN
    rsdone = 1b;
ELSE
    rsdone = 0b;
END IF;

IF (addro[6..0] = 1101111b) THEN
    spdone = 1b;
ELSE
    spdone = 0b;
END IF;

" Read the status then write to the FIFO
IF ((sb[8..0] = 00000000b) AND (/cs_sc) AND (/rdb)) THEN
    dat_sel=1b;
    out_en=1b;

    ELSIF sb[8..0] = 000000110b THEN
        [cnt3a_en,dat_sel,out_en,wen] = 1111b;
        ctrlreg_out;

    ELSIF sb[8..0] = 000001100b THEN
        cnt5_en = 1b;
        gen_cs0; "Generate chip select and read for the PIFS
        out_4byte; "Output 4 byte of PIFS status to FIFO

    ELSIF sb[8..0] = 000011000b THEN
        cnt3_en = 1b;
        gen_cs(1); "Generate chip select and read for the RSEC"
        out_2byte; "Output 2 byte of RSEC status to FIFO"

    ELSIF sb[8..0] = 000110000b THEN
        cnt3_en = 1b;
        gen_cs(2); "Generate chip select and read for the SP"
        out_2byte; "Output 2 byte of SP status to FIFO"

    ELSIF sb[8..0] = 001100000b THEN
        cnt2_en = 1b;
        gen_cs3; "Generate chip select and read for the FRAME"
        out_1byte; "Output 2 byte of FRAME status to FIFO"

    ELSIF sb[8..0] = 011000000b THEN
        cnt5_en = 1b;
        gen_cs4; "Generate chip select and read for the PACKET"
        out_4byte; "Output 2 byte of PACKET status to FIFO"

"    ELSIF sb[8..0] = 110000000b THEN
"        cnt2_en = 1b;
"        gen_cs5; "Generate chip select and read for the SYSTEM"
"        out_1byte; "Output 1 byte of SYSTEM status to FIFO"
"    END IF;

"16bit address counter

    IF ((/clr) OR (clr_addr)) THEN
        addro = 0;
    ELSIF (addr_en) THEN
        addro = addro .+. 1b;
    ELSE
        addro = addro;
    END IF;

"3 bit counter from 0 to 5 to count 6 clock cycles for long word

    IF ((/clr) OR (clr_cnt5)) THEN
        cnt5 =7;
    ELSIF ((cnt5_en) AND (cnt5 < 5)) THEN
        cnt5 = cnt5 .+. 1b;
    ELSIF ((cnt5_en) AND (cnt5 >= 5)) THEN
        cnt5 = 0;
    ELSE
        cnt5 = cnt5;
    END IF;

"2 bit counter from 0 to 3 to count 4 clock cycles for word

    IF ((/clr) OR (clr_cnt3)) THEN
        cnt3 =7;
    ELSIF ((cnt3_en) AND (cnt3 < 3)) THEN
        cnt3 = cnt3 .+. 1b;
    ELSIF ((cnt3_en) AND (cnt3 >= 3)) THEN
        cnt3 = 0;
    ELSE
        cnt3 = cnt3;
    END IF;

"2 bit counter from 0 to 2 to count 3 clock cycles for byte

```

```

    IF ((/clr) OR (clr_cnt2)) THEN
        cnt2 = 3;
    ELSIF ((cnt2_en) AND (cnt2 < 2)) THEN
        cnt2 = cnt2 .+. 1b;
    ELSIF ((cnt2_en) AND (cnt2 >= 2)) THEN
        cnt2 = 0;
    ELSE
        cnt2 = cnt2;
    END IF;

"2bit counter 0-3 to count 4 clock cycles for ctrlreg output

    IF ((/clr) OR (clr_cnt3a)) THEN
        cnt3a = 0;
    ELSIF ((cnt3a_en) AND (cnt3a <> 7)) THEN
        cnt3a = cnt3a .+. 1b;
    ELSIF ((cnt3a_en) AND (cnt3a = 7)) THEN
        cnt3a = 0;
    ELSE
        cnt3a = cnt3a;
    END IF;

"1bit counter to generate write signal to the FIFO

    IF (/clr) THEN
        cnt1 = 0;
    ELSIF (/cs_fifo AND /we) THEN
        cnt1 = cnt1 .+. 1b;
    ELSE
        cnt1 = cnt1;
    END IF;

data[31..9] = muxout[31..9];
addr[15..0] = addro[15..0];
fr_addr[11..0] = [ctrlreg[11..4],1,1,1,1];
pk_addr[13..0] = ctrlreg[26..13];
"eod = eod1 + eod2;
"clr_addr = clr_addr1 + clr_addr2 + clr_addr3;
clr_addr = clr_addr1 + clr_addr3;

" This state machine generates the chip selects when
" controller wants to access the bus

STATE_MACHINE bus_cs
    STATE_BITS [xb1,ack]
    CLOCKED_BY clk
    RESET_BY /clr;

STATE wait_access [00b]:
    IF ((hold) AND ((sc_done) OR (ack1))) THEN
        GOTO delay;
    ELSE
        GOTO wait_access;
    END IF;

STATE delay [01b]:
    GOTO cs_decode;

STATE cs_decode [11b]:
    IF (/hold) THEN
        GOTO wait_access;
    ELSE
        GOTO cs_decode;

    IF addr[17..13]=00000b THEN
        cs_sc = 0b; "sc chip control register
    ELSIF addr[17..13]=00001b THEN
        cs_fifo=0b; " write enable to the status fifo
    ELSIF addr[17..14]=0001b THEN
        cs_a[0]=0b; " pifs chip
    ELSIF addr[17..14]=0010b THEN
        cs_a[1]=0b; " rsec chip register
    ELSIF addr[17..14]=0011b THEN
        cs_a[5]=0b; " rsec memory
    ELSIF addr[17..14]=0100b THEN
        cs_a[2]=0b; " sp register
    ELSIF addr[17..14]=0101b THEN
        cs_a[3]=0b; " sp frame status
    ELSIF addr[17..15]=011b THEN
        cs_a[4]=0b; " sp packet status
    ELSIF addr[17..16]=10b THEN
        cs_a[6]=0b; " sp memory space
    END IF;
END IF;
END bus_cs;

" This state machine loads the control register and collects
" the status

STATE_MACHINE sc

```



```

STATE_BITS sb[8..0]
CLOCKED_BY clk
RESET_BY /clr;

STATE idle [000000000b]:
    IF (/cs_sc AND /we AND /addr[0]) THEN
        ctrlreg[26..0] = data[26..0];
        GOTO if_ready;
    ELSE
        ctrlreg[26..0] = ctrlreg[26..0];
        GOTO idle;
    END IF;
STATE if_ready [000000010b]:
    IF ((aff) AND (/ack)) THEN
        cnt3a_en = 1b;
        GOTO out_ctrlreg;
    ELSE
        GOTO if_ready;
    END IF;
STATE out_ctrlreg [000000110b]:
    IF cnt3a = 7 THEN
        clr_cnt3a = 1b;
        clr_addr3 = 1b;
        GOTO check_pifs;
    ELSE
        GOTO out_ctrlreg;
    END IF;
STATE check_pifs [000000100b]:
    IF ctrlreg[0] THEN
        GOTO collect_pifs;
    ELSE
        GOTO check_rsec;
    END IF;
STATE collect_pifs [000001100b]:
    " IF ((addro[4..0]=11111b) AND (cnt5=5)) THEN
        IF ((fsdone) AND (cnt5=5)) THEN
            clr_cnt5 = 1b;
            clr_addr1 = 1b;
            eod1 = 1b;
            "
            eod = 1b;
            GOTO check_rsec;
        ELSIF ((cnt5=5) AND (hold OR /aff)) THEN
            GOTO pend_state0;
        ELSE
            GOTO collect_pifs;
        END IF;
STATE check_rsec [000001000b]:
    IF ctrlreg[1] THEN
        GOTO collect_rsec;
    ELSE
        GOTO check_sp;
    END IF;
STATE collect_rsec [000011000b]:
    " IF ((addro[5..0]=100111b) AND (cnt3=3)) THEN
        IF ((rsdone) AND (cnt3=3)) THEN
            clr_cnt3 = 1b;
            clr_addr1 = 1b;
            eod1 = 1b;
            "
            eod = 1b;
            GOTO check_sp;
        ELSIF ((cnt3=3) AND (hold OR /aff)) THEN
            GOTO pend_statel;
        ELSE
            GOTO collect_rsec;
        END IF ;
STATE check_sp [000010000b]:
    IF ctrlreg[2] THEN
        GOTO collect_sp;
    ELSE
        GOTO check_frm;
    END IF;
STATE collect_sp [000110000b]:
    " IF ((addro[6..0]=1101111b) AND (cnt3=3)) THEN
        IF ((spdone) AND (cnt3=3)) THEN
            clr_cnt3 = 1b;
            "
            clr_addr2 = 1b;
            clr_addr1 = 1b;
            eod2 = 1b;
            "
            eod = 1b;
            GOTO check_frm;
        ELSIF ((cnt3=3) AND (hold OR /aff)) THEN
            GOTO pend_state2;
        ELSE
            GOTO collect_sp;
        END IF;
STATE check_frm [000100000b]:
    IF ctrlreg[3] THEN
        GOTO collect_frm;
    ELSE
        GOTO check_pkt;
    END IF;
STATE collect_frm [001100000b]:

```

```

        IF (fr_eq AND (cnt2=2)) THEN
            clr_cnt2 = 1b;
            clr_addr2 = 1b;
            clr_addr1 = 1b;
            eod2 = 1b;
        "
        eod = 1b;
        GOTO check_pkt;
        ELSIF ((cnt2=2) AND (hold OR /aff)) THEN
            GOTO pend_state3;
        ELSE
            GOTO collect_frm;
        END IF;
STATE check_pkt [001000000b]:
    IF ctrlreg[12] THEN
        GOTO collect_pkt;
    ELSE
        GOTO check_sys ;
        GOTO idle ;
    END IF;
STATE collect_pkt [011000000b]:
    IF (/pk_eq AND (cnt5=5)) THEN
        clr_cnt5 = 1b;
        clr_addr2 = 1b;
        clr_addr1 = 1b;
        eod2 = 1b;
    "
    eod = 1b;
    GOTO check_sys;
    GOTO idle;
    ELSIF ((cnt5=5) AND (hold OR /aff)) THEN
        GOTO pend_state4;
    ELSE
        GOTO collect_pkt;
    END IF;
"STATE check_sys [010000000b]:
"    IF ctrlreg[25] = 1 THEN
"        GOTO collect_sys;
"    ELSE
"        GOTO idle;
"    END IF;
"STATE collect_sys [110000000b]:
"    IF ((addro[1..0] = 11b) AND (cnt2=2)) THEN
"        clr_cnt2 = 1b;
"        clr_addr2 = 1b;
"        eod2 = 1b;
"        GOTO idle;
"    ELSIF ((cnt2=2) AND (hold OR /aff)) THEN
"        GOTO pend_state5;
"    ELSE
"        GOTO collect_sys;
"    END IF;
STATE pend_state0 [100000000b]:
    IF (hold OR /aff) THEN
        GOTO pend_state0;
    ELSE
        GOTO collect_pifs;
    END IF;
STATE pend_state1 [100000001b]:
    IF (hold OR /aff) THEN
        GOTO pend_state1;
    ELSE
        GOTO collect_rsec;
    END IF;
STATE pend_state2 [100000011b]:
    IF (hold OR /aff) THEN
        GOTO pend_state2;
    ELSE
        GOTO collect_sp;
    END IF;

STATE pend_state3 [100000010b]:
    IF (hold OR /aff) THEN
        GOTO pend_state3;
    ELSE
        GOTO collect_frm;
    END IF;

STATE pend_state4 [100000110b]:
    IF (hold OR /aff) THEN
        GOTO pend_state4;
    ELSE
        GOTO collect_pkt;
    END IF;
"STATE pend_state5 [100000100b]:
"    IF (hold OR /aff) THEN
"        GOTO pend_state5;
"    ELSE
"        GOTO collect_sys;
"    END IF;
END sc;

```

"4 bit counter to divide clk by 16 to generate adcck

```

    IF (div <> 15) THEN
        div = div .+. 1b;
    ELSE
        div = 0;
    END IF;

```

adcck = div[3];

" state machine to generate chip select and address to adc

```

STATE_MACHINE atod
    STATE_BITS [adca[1..0],adccs];

```

```

STATE dostart    [001b]:
    GOTO read_zero;

```

```

STATE read_zero  [000b]:
    adcreg[15..8] = adcd;
    GOTO refresh;

```

```

STATE refresh    [011b]:
    GOTO read_one;

```

```

STATE read_one   [010b]:
    adcreg[7..0] = adcd;
    GOTO dostart;

```

END atod;

B.4 PCI CONFIGURATION ROM DATA LISTING

This is the data for the AT24C02, the serial EEPROM used by the V962PBC to load PCI configuration information from. This is the Intel 32-bit Hexidecimal Object (Hex-32) file, RLP.HEX.

```

:0200000020000FC
:10000000FFFFFFFF00008000000000FF0000000075
:1000100000000000000000000000000000000000E0
:1000200000000000000000000000000000000000D0
:1000300000000000000000000000000000000000BF
:10004000130000000000000000C0000000C00020083
:1000500000000000000000000000000000000000A0
:100060000000000000000000000000000000000090
:10007000F0F000000000000000E0000000000000C0
:00000001FF

```

APPENDIX C

PARTS LIST

This appendix contains a detailed parts list for the RLP.

APPENDIX D

ENGINEERING CHANGE ORDERS

This appendix includes photocopies of all released Engineering Change Orders (ECOs) issued against the assembly revision at the time of document release, as listed in Table D-1.

Table D-1. Directory of Engineering Change Orders

ECO Number	Issue	Date

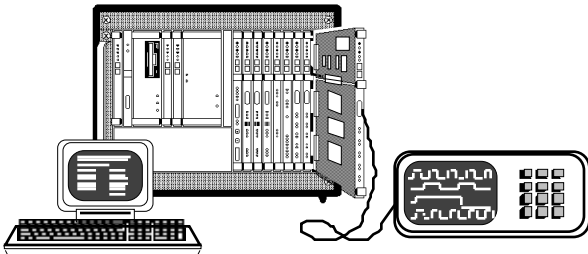
APPENDIX E

TEST PROCEDURES

E.1 INTRODUCTION

This section will be written after experience with the hardware has been obtained. The following is example/template material.

Code 521
Integration and Testing
Group
Card-Level
Test Procedures
Acceptance



This form is to be used to verify and document that test procedures have been used and accepted by Integration and Testing (I&T), the responsible design engineer, and Technical Publications. The actual test procedures, which comprise Appendix E in the associated Hardware Definition Document (HDD), will not be incorporated into the document without completion of this form. This form will be reproduced in the final version of the HDD; please write legibly.

Test procedure acceptance consists of the following three stages:

Step 1				
Date	Test Conductor (Initials)		Responsible Engineer (Initials)	Description
				Test procedures have been used by the Test Conductor with the assistance of the responsible engineer to test one card.

Step 2		
Date	Test Conductor (Initials)	Description
		Test procedures have been used by the Test Conductor to test one card <i>without the assistance</i> of the responsible engineer.

Step 3				
Date	I&T Supervisor (Initials)		Technical Publications (Initials)	Description
				Following review with Test Conductor, I&T Supervisor approves test procedures and submits to Technical Publications.

NOTES:

Please contact the Integration and Testing Supervisor for questions regarding these procedures.

Figure E-1. Integration and Testing Card-Level Test Procedures Acceptance Form

E.1.1 EXPECTED TEST RESULTS

E.1.2 PROCEDURE OUTLINE**Table E-1. Memory Map with Chip Selects**

Chip Select	Address (Hex)	Device/Function
CS0		
CS1		
CS2		
CS3		
CS4		
CS5		
CS6		
CS7		
CS8		
CS9		
CS10		
CS11		

E.1.3 TESTING REQUIREMENTS**Figure E-2. Example Start File****E.1.3.1 Memory Tests****E.1.3.2 Register Tests****E.1.3.3 Chip Selects****E.1.3.4 Resets**

E.1.3.5 Data Flow**E.1.3.6 Rate Testing****E.1.4 APPLICABLE DOCUMENTS****E.2 CARD TEST PROCEDURES****E.2.1 STEP 1 TO STEP N****E.3 TESTPOINT SIGNALS****Table E-2. Example Testpoint**

Pin Number	Signal
1	GND
2	D<1>
3	D<3>
4	D<5>
5	D<7>
6	CLK8X
7	CLK4XINV
8	SELBTD
9	EOF
10	GND
11	GND
12	Used for clock signal
13	EODS
14	EOSYNC
15	CLK4X
16	PCLK
17	D<6>
18	D<4>
19	D<2>
20	D<0>

Figure E-5. Example of a Diagram that Illustrates Testpoints